**Entwicklung einer einfachen Kunden und Vertragsverwaltung**

**IPA - Bericht**

**Huanga IT Solutions AG**  
Sägereistrasse 21  
8152 Opfikon

|  |  |
| --- | --- |
| Autor: | Aaron Schreiber |
| Datum: | 25.04.2023 |

Inhaltsverzeichnis

[Inhaltsverzeichnis 1](#_Toc133841063)

[Vorwort 2](#_Toc133841064)

[Informationen zum Dokument 3](#_Toc133841065)

[1. Erster Teil des IPA-Bericht: Umfeld und Ablauf 3](#_Toc133841066)

[1.1 Titel der Arbeit 3](#_Toc133841067)

[1.2 Ausgangslage 3](#_Toc133841068)

[1.3 Auftrag 3](#_Toc133841069)

[1.4 Detaillierte Aufgabenstellung 3](#_Toc133841070)

[1.5 Mittel und Methoden 5](#_Toc133841071)

[1.6 Vorkenntnisse 5](#_Toc133841072)

[1.7 Vorarbeit 5](#_Toc133841073)

[1.8 Neue Lerninhalte 5](#_Toc133841074)

[1.9 Arbeit in den Letzten 6 Monaten 5](#_Toc133841075)

[1.10 Verfeinerte Aufgabenstellung 5](#_Toc133841076)

[1.11 Hardware 6](#_Toc133841077)

[1.12 Software 6](#_Toc133841078)

[1.13 Projektaufbauorganisation (Personen/Rollen/Aufgaben/Verantwortung) 7](#_Toc133841079)

[1.14 Firmenstandards 8](#_Toc133841080)

[1.15 Durchführungstermin 8](#_Toc133841081)

[1.16 Projektmethode 9](#_Toc133841082)

[1.17 Dokumentstruktur 10](#_Toc133841083)

[1.17.1 Versionierung 11](#_Toc133841084)

[1.17.2 Backup 11](#_Toc133841085)

[1.18 Zeitplan 12](#_Toc133841086)

[1.19 Arbeitsjournal 13](#_Toc133841087)

[2. Zweiter Teil des IPA-Berichtes: Projekt 13](#_Toc133841088)

[2.1 Kurzfassung des IPA-Berichts 13](#_Toc133841089)

[2.1.1 Ausgangslage 13](#_Toc133841090)

[2.1.2 Umsetzung 13](#_Toc133841091)

[2.1.3 Ergebnis 13](#_Toc133841092)

[2.2 Informieren 13](#_Toc133841093)

[2.2.1 Kundengespräch 13](#_Toc133841094)

[2.2.2 MVC-Framework 14](#_Toc133841095)

[2.2.2.1 Erläuterung 14](#_Toc133841096)

[2.2.2.2 Entwurfsmuster (Design Patterns) 14](#_Toc133841097)

[2.2.3 Kriterienkatalog Studieren 15](#_Toc133841098)

[2.3 Planen 15](#_Toc133841099)

[2.3.1 Arbeitsschritte für die Applikation definieren 15](#_Toc133841100)

[2.3.2 Komplexe Features Konzepte Planen 16](#_Toc133841101)

[2.3.3 Meilensteine Definieren 17](#_Toc133841102)

[2.3.4 ERM Erstellen 18](#_Toc133841103)

[2.3.5 Designkonzept 18](#_Toc133841104)

[2.3.6 Testkonzept Erstellen 20](#_Toc133841105)

[2.4 Entscheiden 21](#_Toc133841106)

[2.4.1 Komplexe Feature Konzepte Auswerten 21](#_Toc133841107)

[2.5 Realisieren 22](#_Toc133841108)

[2.5.1 Framework Umgebung 22](#_Toc133841109)

[2.5.2 AdminLTE Partial Viewkonzept bereitstellen 23](#_Toc133841110)

[2.5.3 Persistente Daten 23](#_Toc133841111)

[2.5.3.1 DB-Benutzer erstellen 23](#_Toc133841112)

[2.5.3.2 Datenbank erstellen 23](#_Toc133841113)

[2.5.3.3 Tabelle erstellen 23](#_Toc133841114)

[2.5.4 CRUDs erstellen 23](#_Toc133841115)

[2.5.4.1 Entitäten Klassen erstellen 23](#_Toc133841116)

[2.5.4.2 Repository Klassen erstellen 23](#_Toc133841117)

[2.5.4.3 Controller Klasse erstellen 23](#_Toc133841118)

[2.5.4.4 View Klassen erstellen 23](#_Toc133841119)

[2.5.4.5 User Login Funktionen hinzufügen 23](#_Toc133841120)

[2.5.5 Formular Validiedrungskonzept einbauen 23](#_Toc133841121)

[2.5.6 Exception Handling Konzept einbauen 23](#_Toc133841122)

[2.5.7 Statistiken einfügen 24](#_Toc133841123)

[2.6 Kontrollieren 24](#_Toc133841124)

[2.6.1 Testprotokol 24](#_Toc133841125)

[2.7 Auswerten 24](#_Toc133841126)

[2.7.1 Selbstreflexion und Fazit 24](#_Toc133841127)

[3 Anhang 24](#_Toc133841128)

[3.1 Quellenverzeichnis 24](#_Toc133841129)

[3.2 Abbildungsverzichnis 24](#_Toc133841130)

[3.3 Tabellenverzeichnis 24](#_Toc133841131)

[3.4 Abkürzungsverzeichnis 24](#_Toc133841132)

[3.5 Glossar 24](#_Toc133841133)

[3.6 Quellcode 24](#_Toc133841134)

Vorwort

Dies ist die Abschlussarbeit von Aaron Schreiber in der Firma Huanga IT Solutions AG in Opfikon ZH. Sie wird in Form einer IPA durchgeführt. Die Arbeit ist in zwei Hauptteile aufgeteilt (Umfeld und Ablauf) und Projekt. Am Schluss befindet sich noch der Anhang. Es wird als ein PDF mit gemeinsam Verzeichnis eingereicht.

Informationen zum Dokument

|  |  |
| --- | --- |
| Dokumenttitel : | IPA-Bericht |
| Autor : | Aaron Schreiber |
| Dateiname | IPA\_Bericht\_Aaron\_Schreiber.docx |

Tabelle 1: Dokument Information

1. Erster Teil des IPA-Bericht: Umfeld und Ablauf

1.1 Titel der Arbeit

Entwicklung einer einfachen Kunden und Vertragsverwaltung

1.2 Ausgangslage

Ein neues Startup mit dem Namen Caskade AG ( https://caskade.ch/ ) braucht eine Kunden- und Vertragsverwaltung. Das Startup kauft im Namen des Kunden Rum-Fässer und lagert diese dann für mehrere Jahre. Am Ende der Lagerung werden die Fässer dann normalerweise an Abfüllfirmen weiterverkauft. Die Kunden partizipieren am Gewinn, welcher zwischen Einkauf und Verkauf entsteht. Weiter bietet Caskade nach dem gleichen Prinzip Rinder in Argentinien an. Das Startup braucht eine eigene Software, welche genau nur die minimalen Bedürfnisse abdeckt.

1.3 Auftrag

Der Lehrling wird an der IPA ein Kunde und verwaltungssoftwareschreiben die für das Caskade AG Startup verwendet wird.

1.4 Detaillierte Aufgabenstellung

Bis jetzt verwaltet das Startup Caskade AG alle ihre Kunden mit Papierverträgen. Dies wird nun mühsam, da z.B. nicht einfach Statistiken erstellt werden können. Der Lehrling muss nun anhand der Papierverträge ein ERM erstellen und daraus dann die Software schreiben. Das Startup führt mit all seinen Kunden ein Verkaufsprotokoll. Dieses soll auch in der Software abgelegt werden können. Funktional verlangt wird:

* Einfache Benutzerverwaltung (ohne Rechteverwaltung; alle Benutzer haben gleiche Rechte)
  + Benutzer erstellen, bearbeiten, löschen
  + Passwort reset (Sowohl als eingeloggter Benutzer das Passwort von einem anderen Benutzer reseten als auch auf dem Login-Bildschirm den Reset vom eigenen Passwort veranlassen.)
* Authentisierung der Benutzer mittels E-Mailadresse (Username) und Passwort
* Produkte-Stammdatenverwaltung
* Kundenverwaltung
  + Kunden erstellen, bearbeiten, deaktivieren und löschen
  + Detailansicht des Kunden mit Vertragsübersicht
* Vertragsverwaltung
  + Verträge erstellen, bearbeiten, deaktivieren
  + Dateiupload pro Vertrag (Verkaufsprotokoll)
* Statistiken bezüglich den Verträgen. Im Detail:
  + Welche Verträge laufen in einem bestimmten Monat aus. (Benutzer wählt Jahr und Monat aus)
  + Welches Umsatzvolumen haben wir im gesamten. (Benutzer wählt Jahr aus, oder "gesamt")
  + Liste aller Verträge und deren Details
  + Für das Design der Benutzeroberfläche soll das Bootstrap basierende Admin-Theme "AdminLTE" verwendet werden.

Da der Lehrling nur die zweijährige Lehre macht, konnte er sich bis jetzt in keinem öffentlichen PHP-Framework vertiefen. Die Software soll darum in unserem MVC-Lern-Framework erstellt werden. Das Lern-Framework bietet nur die minimale Aufteilung des Codes in Model, View und Controller. Dazu stellt es ein paar grundlegende Komponenten wie Router zur Verfügung. Der gesamte Rest wie z.B. Datenfilter muss von Hand aus-programmiert werden.

Sicherheitsvorgaben: Das Passwort gilt als sensitiv und muss darum als Hash gespeichert werden. Weiter muss sichergestellt werden, dass keine Daten angeschaut werden können, ohne eingelogged zu sein. Beim Passwort-Reset muss darauf geachtet werden, dass Hacker nicht herausfinden können, welche Benutzernamen/E-Mailadressen schon existieren. Es wird erwartet, dass der Code sauber gemäss phpDoc Standard (docblock) dokumentiert wird. Ein weiterer Standard, welcher eingehalten werden muss ist der Coding Standard https://www.php-fig.org/per/coding-style/ Der Code muss in unserem Versionierungssystem gitlab abgelegt werden. Auch die Dokumentation muss dort abgelegt werden.

1.5 Mittel und Methoden

- PHP (Minimales MVC Framework) - AdminLTE (Bootstrap basierendes CSS Framework)

1.6 Vorkenntnisse

Der Lehrling macht die verkürzte 2 jährige Lehre. Er hat vor 2 Jahren mit null IT Kenntnissen angefangen und um ersten Halbjahr nur an Websites mitgearbeitet. Parallel zur Schule kamen dann einzelne kleinere Softwareentwicklungsprojekte hinzu. Seit Anfangs 2023 werden auch Themen wie Design Patterns und Laminas Framework angeschaut. Wegen der kurzen Lehrzeit konnten diese Themen aber nicht sehr stark vertieft werden.

1.7 Vorarbeit

Gespräch(e) mit Kunde / Fachvorgesetzter zur Klärung des genauen Umfangs der Software.

1.8 Neue Lerninhalte

Der Lehrling hat noch nie das AdminLTE verwendet, wohl aber schon sehr viel mit purem CSS gemacht. Ich erwarte da keine Probleme.

1.9 Arbeit in den Letzten 6 Monaten

Verschiedene kleinere Projekte mit PHP oder PHP basierenden Frameworks wie (Laminas, WordPress, Contao, eigenes MVC-Framework)

1.10 Verfeinerte Aufgabenstellung

Die Detaillierte Aufgabenstellung, welche von der verantwortlichen Fachkraft geschrieben wurde, konnte in folgende Teilaufgaben unterteilt werden:

**Entwurf - SW-Architektur**

Es wurde eine gute Klassenstruktur gewählt. Die Klassen haben schlanke, wohldefinierte Schnittstellen, welche auch dokumentiert sind (im Quellcode oder anderswo). Die Klassen sind in sinnvollen Modulen untergebracht.

**Versionsverwaltung mit Verwaltungs-SW**

Es wird ein Versionsverwaltungssystem für Daten, Programme und Dokumente verwendet. Das System erlaubt Aussagen zum Zustand jeder Version. Es wird ein Backupsystem für Daten, Programme und Dokumente verwendet.

Ein Restore aller relevanten Versionen inkl. passenden Testdaten während der ganzen Facharbeit ist jederzeit möglich. Das Backup ist mindestens tagesaktuell.

**Professionelle Web-Programmierung**

* Klares Programm-Design
* Session-Control
* Lesbarer Code (Variablen- und Funktionsnamen, Struktur).
* Aussagekräftiger Kommentar
* Korrekter Einsatz der Sprachelemente
* Wartbar dank sinnvollen Parametern

**Vollständiges ERM bzw. Datenmodell**

Alle Entitäten und Beziehungen sind korrekt dargestellt.Alle Assoziationstypen (1, c, m, mc) sind korrekt eingetragen.Alle Primärschlüssel und Fremdschlüssel sind als solche erkenntlich, also entsprechend markiert bzw. bezeichnet.Alle Attribute listen sind vollständig, die Datentypen aller Attribute sind angegeben.

**Codierung: Fehlerbehandlung**

Es wurde eine Strategie zur Fehlerbehandlung eingeführt und im ganzen Code konsistent verwendet.

**GUI-Design (Fokus: Benutzerfreundlichkeit über die ganze Applikation)**

Masken in der richtigen Reihenfolge (Applikation bildet den Prozess/Workflow richtig ab). Ist das Design durchgängig (gleiche Elemente am gleichen Platz) Kann der Benutzer bei Fehleingaben zurück navigieren (oder ist für den Benutzer transparent dargestellt, warum dies nicht möglich ist)?

**Plausibilisierung der Benutzer-Eingaben**

Alle Eingabefelder werden überprüft. Es ist eindeutig gekennzeichnet, welche Felder Pflichtfelder sind. Für den Benutzer ist ersichtlich, welche Wertebereiche zulässig sind. Findet die Plausibilisierung eine Fehleingabe, so wird der Benutzer mit konkreten Hinweisen geführt, das entsprechende Feld wird aktiviert.

1.11 Hardware

|  |  |
| --- | --- |
| **Arbeitsgerät (Arbeits Laptop)** | DELL Latitude 5490 |
| **Dokumentationsgerät (Persönlicher Laptop)** | MacBookPro 18,3 |

Tabelle 2: Hardware

1.12 Software

|  |  |
| --- | --- |
| **Entwicklung** | * Microsoft Visual Studio * PhpMyAdmin * Diagramms.net * Microsoft Word * Microsoft Excel |
| **Programmiersprachen** | * PHP * HTML * CSS * SQL * Javascript/JQuery |

Tabelle 3: Software

1.13 Projektaufbauorganisation (Personen/Rollen/Aufgaben/Verantwortung)

|  |  |
| --- | --- |
| Rolle | Kandidat |
| Vorname Nachname  Adresse | Aaron Schreiber Burgstrasse 54 8610 Uster |
| Aufgabe / Verantwortung | Ausführender der IPA |
|  |  |
| Rolle | Verantwortliche Fachkraft |
| Vorname Nachname | Raffael Luthiger |
| Aufgabe / Verantwortung | Betreuung, Bewertung der IPA |
|  |  |
| Rolle | Berufsbildner |
| Vorname Nachname | Christian Walder |
| Aufgabe / Verantwortung | Anmeldung des Kandidaten |
|  |  |
| Rolle | Hauptexperte |
| Vorname Nachname | Marco Lupi |
| Aufgabe / Verantwortung | Bewertung der IPA |
|  |  |
| Rolle | Nebenexperte |
| Vorname Nachname | Michael Stolz |
| Aufgabe / Verantwortung | Bewertung der IPA |
|  |  |
| Betrieb (Durchführungsort) Adresse | Huanga IT Solutions AG Sägereistrasse 21 Opfikon 8152 |

Tabelle 4: Personen und Rollen

1.14 Firmenstandards

* Die Applikation sollte mit docblock aufgebaut werden.
* K&R Einrückungsstyle sollte für das Projekt benutzt werden.
* Das Programm sollte auf GitLab versioniert werden.

1.15 Durchführungstermin

|  |  |
| --- | --- |
| Termin | |
| Starttermin | 25.04.2023, 7:30 |
| 1 Experten Besuch | 27.04.2023, 13.00 |
| 2 Experten Besuch | 05.05.2023 |
| Abgabetermin | 09.05.2023 |
| Präsentation | 23.05.2023 |

Tabelle 5: Termine

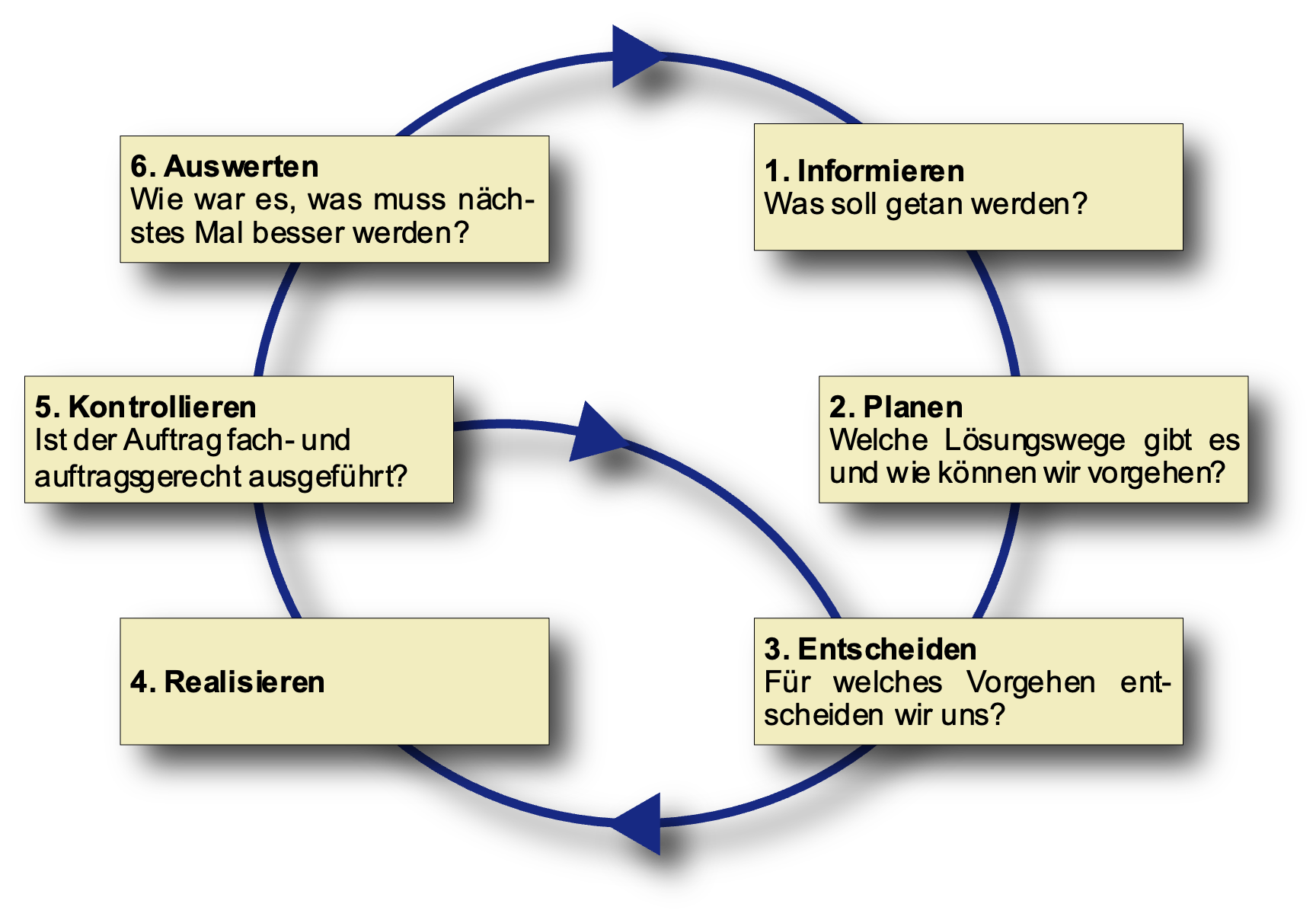
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Kalenderwoche 17 | | | | | |
|  | **24.04.23** | **25.04.23** | **26.04.23** | **27.04.23** | **28.04.23** |
| **Vormittag** | Frei | IPA | IPA | IPA | IPA |
| **Nachmittag** | Frei | IPA | IPA | IPA | IPA |
| **Abend** |  | Schule |  | Schule |  |
| Kalenderwoche 18 | | | | | |
|  | **01.05.23** | **02.05.23** | **03.05.23** | **04.05.23** | **05.05.23** |
| **Vormittag** | Frei | IPA | IPA | IPA | IPA |
| **Nachmittag** | Frei | IPA | IPA | IPA | IPA |
| **Abend** |  | Schule |  | Schule |  |
| Kalenderwoche 19 | | | | | |
|  | **01.05.23** | **02.05.23** | **03.05.23** | **04.05.23** | **05.05.23** |
| **Vormittag** | IPA | IPA |  |  |  |
| **Nachmittag** | IPA | IPA |  |  |  |
| **Abend** |  | Schule |  |  |  |

Tabelle 6: Durchführungstermine

1.16 Projektmethode

Die Projektmethode IPERKA wurde gewählt da diese mir schon bekannt war und weil es sich für kleineren Projekten sehr empfiehlt.

Anhand folgender Fragen werde ich IPERKA nutzen:

* **Informieren**  
  Was sind die Vorgaben für das Produkt?  
  Wie lautet der Auftrag?
* **Planen**  
  Ziele definieren  
  Lösungswege bestimmen

Konzepte erstellen

* **Entscheiden**

Vorgehen festlegen

* **Realisieren**  
  Erarbeitung der geplanten Konzepte
* **Kontrollieren**

Testen des Produktes

* **Auswerten**

Wie war die Umsetzung?

Was könnte besser gemacht werden?

Abbildung 1: IPERKA Ablauf

1.17 Dokumentstruktur

Text

Description automatically generatedDie Struktur des Projekts ist logisch und übersichtlich.

Abbildung 2: Dokumentenstrukturierung

In *classes* sind alle framework spezifischen Klassen.

*Controllers* ist das Bindeglied zwischen *models* und *views*. *Controllers* holt diese Daten bei *models* und verarbeitet diese um sie danach an *views* weiterzugeben.

*Documentation* ist der Ort in dem der IPA-Bericht und andere Dokumente, die zur Evaluierung des Projekts wichtig sind, gespeichert werden.

*Models* ist die Schnittstelle zu den Daten.

*Views* zeigt die Daten von *models*, die von *controllers* bereitgestellt werden.

1.17.1 Versionierung

IPA-Dokumentation Dateiname: IPA\_Bericht\_Aaron\_Schreiber.docx

|  |  |  |
| --- | --- | --- |
| Version | Datum | Bemerkung |
| 01 | 25.04.2023 | Version hochgeladen |
| 02 | 26.04.2023 | Version hochgeladen |
| 03 | 27.04.2023 | Version hochgeladen |
| 04 | 28.04.2023 |  |
| 05 | 02.05.2023 |  |
| 06 | 03.05.2023 |  |
| 07 | 04.05.2023 |  |
| 08 | 05.05.2023 |  |
| 09 | 08.05.2023 |  |
| 10 | 09.05.2023 |  |

Tabelle 7: Versionierung IPA-Bericht

1.17.2 Backup

Alle Daten werden auf dem firmeneigenen GitLab Repository gespeichert.

So entsteht jeden Tag eine neue Version der Files.

1.18 Zeitplan

Abbildung 3: Zeitplan IPA-Bericht

1.19 Arbeitsjournal

|  |  |
| --- | --- |
| Datum |  |
| Arbeit |  |
| Erfolge / Misserfolge |  |
| Aufgetretene Probleme |  |
| Allfällige Hilfestellung (+ Begründung) |  |
| Überzeit |  |
| Ungeplante Arbeit |  |
| Reflexion |  |

2. Zweiter Teil des IPA-Berichtes: Projekt

2.1 Kurzfassung des IPA-Berichts

2.1.1 Ausgangslage

Für das Startup Caskade, wird im Verlauf der IPA eine Webapplikation erstellt. Für die Applikation werden mehrere CRUDs erstellt. Mit diesen kann man Benutzer, Kunden, Produkte und Verträge bearbeiten.

2.1.2 Umsetzung

Die Applikation soll mit PHP, SQL, HTML, JS erstellt werden. Zuerst wird eine Datenbank erstellt. Dann werden die Entitäten erstellt und deren Repositories. Nachher erstellt man die Controller und zu guter Letzt die Views.

2.1.3 Ergebnis

Es soll eine Vertragsverwaltungssoftware erstellt werden, mit der man neue Verträge erstellen kann. Zudem kann man Kunden verwalten und diese erstellen, bearbeiten, löschen und deaktivieren.

2.2 Informieren

2.2.1 Kundengespräch

Das Kundengespräch soll dazu dienen, die Anforderungen an die Software genau zu evaluieren. Ich traf mich mit Raffael Luthiger ein paar Tage vor der IPA, um all meine offenen Fragen mit ihm zu klären. Im speziellen wollte ich mit ihm alle Kleinigkeiten durchgehen, die in der Aufgabenstellung nicht genau definiert waren. Danach trafen wir uns ein zweites Mal am 25.04.23 um zu spezifizieren, wie die Verträge aufgebaut werden sollen. Durch das Gespräch habe ich erfahren, dass ein oder zwei Kunden zu einem Vertrag gehören. Dasselbe trifft auch für die Produkte zu also auch eins oder zwei Produkte pro Vertrag. Das ist ausschlaggebend für das CRUD der Verträge. Auch wurde festgelegt das der Legitimationsnachweis Teil des Vertrags nicht benötigt wird.

2.2.2 MVC-Framework

2.2.2.1 Erläuterung

Das MVC-Framework ist eine firmeninterne Applikation, die von einem anderen Lehrling erstellt und gewartet wurde. Das MVC-Framework ist simple und sollte schnell erklären, wie architekturale Entwurfsmuster funktionieren sollen.

2.2.2.2 Entwurfsmuster (Design Patterns)

Es gibt einige Entwurfsmuster, die im MVC-Framework verwendet werden. Einige werden ganz eingesetzt. Einige benutzen nur Ideen eines Musters aber implementieren es nicht ganz. Im untenstehenden Teil werden diese aufgezählt und es wird erläutert, wie diese verwendet werden.

**MVC**

Wie der Name schon sagt, wird in diesem Framework das architekturale Entwurfs Muster des MVC verwendet. Das Problem, das wir mit diesem Entwurfsmuster lösen wollen, ist die Separation von Code und die nicht vorhandene oder nur schwache Erweiterbarkeit von Code. Dies lösen wir mit den folgenden drei Teilbereichen:

* Das Model ist die Schnittstelle zu Daten. Es wird verwendet, um Daten zu holen und zu schreiben.
* Die View ist das, was der Endbenutzer sieht. Die Daten werden vom Controller bereitgestellt und dieser holt die Daten vom Model.
* Wie schon erklärt ist der Controller der Mittelsmann zwischen der View und dem Model. Er verarbeitet requests, fragt Daten vom Model ab und verarbeitet diese und manipuliert die View.

**Registry**

Die Registry macht bestimmte Daten für alle Teile der Applikation verfügbar. In unserem Framework wird die Registry dazu verwendet um Variablen für alle Teile des Frameworks zu setzten. Das heisst, dass ich mit der Registry eine Variable setzten kann und nachher diese überall im Programm benutzen kann. Zudem kann man Variablen auch entfernen. Das ist praktisch, wenn man an einem Punkt im Programm eine Variable überschreiben will um ihr einen anderen Wert zu geben.

In einem späteren Abschnitt werde ich noch auf die anderen Design Patterns eingehen, die ich in meinem Programm verwenden werde.

2.2.3 Kriterienkatalog Studieren

In diesem Abschnitt habe ich mir den Kriterien Katalog ausgedruckt, gelesen und mir alles Wichtige angestrichen. Damit ich alle Prüfungskriterien kenne und anhand dieser mein Projekt strukturieren kann. Somit kann ich auch reflektieren was schon erledigt ist und was eventuell nochmals angeschaut werden sollte.

2.3 Planen

Die Planung erfolgt durch das Festlegen aller Arbeitsschritte, die für die Applikation benötigt werden, das konzeptuelle Erarbeiten von Features, die später ausprogrammiert werden müssen und die Definition von Meilensteinen. Die Planung ist im Zeitplan ersichtlich und logisch anhand der Tätigkeiten strukturiert.

2.3.1 Arbeitsschritte für die Applikation definieren

Die Arbeitschritte, die im Endeffekt das zu programmierende Projekt ausmachen, sind die Folgenden:

* AdminLTE Partial View Konzept
* Persistente Daten
  + DB-Benutzer erstellen
  + Datenbank erstellen
  + Tabelle erstellen
* Session handling Konzept einbauen
* CRUDs erstellen
  + Entitäten Klassen erstellen
  + Repository Klassen erstellen
  + Controller Klassen erstellen
  + View Klassen erstellen
* User Login Funktionen hinzufügen
* Formular Validiedrungskonzept einbauen
* Exception Handling Konzept einbauen
* Statistiken einfügen

2.3.2 Komplexe Features Konzepte Planen

Die folgenden komplexen Features müssen vor der eigentlichen Programmierung definiert werden, sodass die Implementation einfacher verläuft:

**Session handling**

**Session Manager Konzept**  
Einen Session Manager erstellen, der jedes Mal schaut, ob die Session gestartet wurde und wenn nicht, diese startet. Dieser Klasse getter und setter für session Werte geben.

**AdminLTE einbinden**

**CDN Konzept**

Mit CDN (Content Delivery Network) kann man AdminLTE einfügen und so einfach implementieren.

**Lokale Dateien Konzept**

Man speichert die Files lokal damit man von einem Ordner direkt darauf zugreifen kann.

**Formularvalidierung**

**Konditional**

Anhand von konditional Statements entscheiden, ob die Daten valid sind, wenn nicht wird anhand der Registry eine Nachricht gespeichert, die dann der View übergeben wird.

**Validierungsklasse**

Durch eine neue Validierungsklasse werden alle Daten im $\_POST durch verschiedene Methoden iteriert, die Mindestlänge oder Maximallänge wird überprüft und dann anhand dieser Daten wird ein Array von Nachrichten zurückgegeben, welche in der View benutzt werden können, um Fehlermeldungen anzuzeigen.

**Exception Handling Konzept**

**Try and Catch**

Für das Exception Handling würde ich Try and catch implementieren, damit ich alle Exceptions fangen kann und diese dem Benutzer ausgeben kann oder sie sonst verarbeiten kann. Hierfür muss ich nur Try and Catch einfügen die Exception Klasse wird von php bereitgestellt. Die Exception Nachricht wird in der Registry gespeichert und dann in der View ausgegeben.

**Session Exceptions**

Handling von Exceptions via Session speichert alle Exceptions in einem Array welches nachher auf jeder View mit einem Foreach ausgegeben wird. Zudem könnte man nicht nur Exceptions sondern auch Bestätigungsnachrichten an die View geben.

**Autocomplete / Multiselect**

Dieses Feature musst ich im nachhinein nochmals Plannen und dokumentieren. Wichtig hier ist das eine zufriedenstellende Funktionalität ahand der Aufgabenstellung geliefert wird aber auch eine die mich Zufreidenstellt. Darum habe ich drei mal dieses Feature überarbeitet um es jetzt in einen zustand zu bringen denn ich Aktzeptabel finde und der sinn macht im Kontext der applikation. Wenn es in der Zukunft mehr als 30-50 benutzer gibt sondern 100+ dann wird es schnell schwierig einen einfachen weg zu einer relation in der Vertrag edit und add methode zu erstellen. Darum habe ich zuerst ein einfaches multiselect plugin verwendet. Habe aber nach input mehrerer benutzer gemerkt das für die Benutzer ein richtiges autocomplete feature gebraucht wird. Die Produkte in diesem zustand mit dem Multiselect field ist ok und kann weiterbestehen. Es macht auch sinn weil ein Vertrag in der Regel nur ein Primären und ein sekundären Käufer hat. Bei produkten kann das aber anders sein darum habe ich das Multiselect field beibehalten.

2.3.3 Meilensteine Definieren

Die Meilensteine sind anhand der Teile von IPERKA gegliedert. Das heisst insgesamt haben wir 6 Meilensteine.

|  |  |
| --- | --- |
| Meilensteine | Erreicht am |
| Meilenstein (Informieren) | 25.04.2023 |
| Meilenstein (Planen) | 26.04.2023 |
| Meilenstein (Entscheiden) | 26.04.2023 |
| Meilenstein (Realisieren) |  |
| Meilenstein (Kontrollieren) |  |
| Meilenstein (Auswerten) |  |

Tabelle 8: Meilensteine

2.3.4 ERM Erstellen

Diagram

Description automatically generated

Abbildung 4: ERM

Im Kunden Gespräch wurde definiert, dass nur ein oder zwei Kunden oder Produkte existieren können. Ich denke aber, dass es auch langfristig mehr Sinn macht, die Daten direkt von Anfang an so zu strukturieren, dass sie auch nach mehreren Jahren die Anforderungen an die Applikation erfüllen. Vielleicht braucht der Kunde in ein paar Jahren 1-3 Produkte für jeden Vertrag und somit machen zwei fixen Spalten in der Tabelle keinen Sinn. Ich denke mit der Zwischentabelle kann man das Problem am besten lösen und gleichzeitig etwas erarbeiten, das später gut gebraucht werden kann.

2.3.5 Designkonzept

Graphical user interface, application, website

Description automatically generatedFür das Design der Applikation verwende ich das AdminLTE Bootstrap Theme. Daher muss ich wenig, bis gar kein CSS schreiben, weil das Theme viele CSS Klassen zur Verfügung stellt. Ich werde für meine Applikation AdminLTE 3 verwenden.

Abbildung 5: AdminLTE

**Graphical user interface, application

Description automatically generated**Für die Applikation sollten alle Formulare als Cards angezeigt werden. Diese sind übersichtlich und haben schon viele Klassen, die ich verwenden kann, um z.b die Spaltengrösse zu definieren. Sie sind mit display: flex; aufgebaut. Das erspart mir Mobil Viewport Styling.

Abbildung 6: AdminLTE 3 Cards

Die Tabellen sollten die Bootstrap Klassen haben. Diese sollten alle gleich gestylt sein und zwei Spalten haben, damit ich alle Daten gut anzeigen kann.

|  |  |
| --- | --- |
| vorname | max |
| nachname | muster |
| strasse | musterstrasse 34 |
| stadt | musterhausen |
| land | musterland |

Abbildung 7: Beispiel Tabelle

Die detaillierten Anzeigen der Daten sollten so angezeigt werden, dass, eine grosse Menge Daten aufgeführt werden kann.

2.3.6 Testkonzept Erstellen

Um die Applikation zu Testen erstelle ich zuerst das Testkonzept. Die eigentlichen Tests finden erst im Kontrollieren Teil von IPERKA statt. Das System das wir Testen ist die Kunden und Vertragsverwaltungssoftware die für diese IPA im MVC framework von Aaron Schreiber erstellt wird. Die tests finden auf verschiedenen Web-browsern statt diese beinhalten:

* Google Chrome (Basiert auf Chromium)
* Edge (Basiert auf Chromium)
* Firefox (Basiert auf Quantum)
* Safari (Basiert auf Webkit)

Die Relevantesten Testfälle und dessen zu erwartenden Ergebnisse sind wie folgt:

|  |  |
| --- | --- |
| Testfall | Erwartetes Ergebnis |
| Alle CRUD-Aktionen und nachher schauen, ob diese in der Datenbank die gewünschten Resultate geben. | Die datenbank sollte anhand der $\_POST daten richtig die daten eingestzen, updaten oder Löschen. |
| Das Routing sollte den richtigen Controller und die richtige Aktion nachweisen. | Die URL spiegelt im MVC Framework das Routing wieder und sollte anhand von dem den Jeweils richtigen Controller und Aktion anzeigen. |
| Die Navigation sollte zur richtigen Seite weiterleiten. | Die weiterleitung von einer Page auf die nächste anhand von Richtigen eingaben. Oder falsche eingaben sollte zur selben seite weitergeleiteten. Natürlich hier wichtig ist die Respektive seite und Controller und Aktion. |
| Der Startpunkt der Applikation sollte beim ersten Mal angezeigt werden. | Die applikation sollte immer einen festen startpunkt im Login haben und wenn man eingeloggt ist auf der Vertrag index seite. |
| Die Login Seite sollte ohne Session respektiv nicht gesetzte Session Schlüssel nicht angezeigt werden. | Wenn kein session mit dem Key Benutzer (in der Applikation auf englisch: „user key has to be set“) existiert dann sollte die Applikation zurück zum Login weiterleiten. Ohne session key benutzer können keine anderen seiten in der Applikation geöffnet werden. |
| Die Validierung sollte anhand der angegeben Validation array die Daten richtig überprüfen | Hierführ entferne ich die HTML5 attribute im browser und überprüfe was die Fehlermeldung ausgeben und ob diese angepasst werden sollten oder ob die daten in der DB richtig aufgebaut sind. |

Der folgende abschnitt wiederspiegelt was nicht getest wird und wieso. Die applikation wird nicht gegen Cyberattacken getestet. Sowie Spitzentest bei dem eine grosse menge an benutzer plötzlich die applikation benutzt. Auch nicht getestet werden Stresstests bei der die Applikation mit Realistischen und unrealistischen weisen Überladen wird um zu sehen wie sie sich verhält.

Zum Testen der Applikation wird Mein Persönlicher Macbook Pro verwendet. Auch für die Mobil view wird der Browser verwendet um diese zu testen. Sodass standard mobil view ports zuverlässig funktionieren. Edge cases wie kleinere geräte mit wenig pixel abdeckung müssen in der Zukunft noch direkt getestet werden aber das ist einfacher wenn die applikation nicht bei mir Lokal auf dem Laptop läuft. Das ist zudem wichtig weil die Darstellung auf dem Browser manchmal ein anderes resultat gibt das eigentlich auf dem smartphone angezeigt wird.

Für die applikation wird Funktionalestesting betrieben bei dem der wichtigste ansatz die Eigentliche funktionalität der Applikation und dessen Features beinhaltet. Zudem wird mit den Verschiedenen Browsern ein Kompatibilitätstest gemacht bei dem Überprüft wird ob die applikation auf denn meist herkömlichen Browsern funktioniert. Zudem wird ein grober viewport test gemacht bei dem getestet wird ob die meisten Standard geräte die applikation richtig anzeigen.

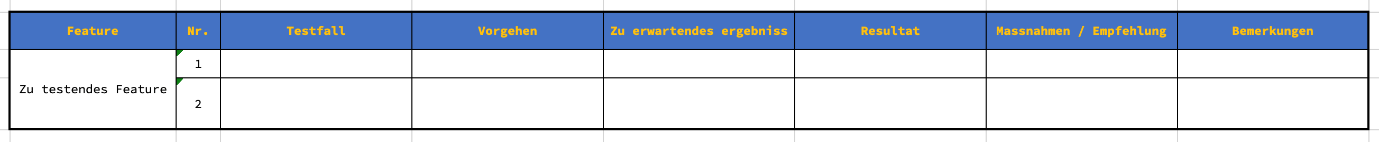
Die ganze Applikation wird getestet, nichts wird übersehen. Für die Tests der Applikation verwende ich Google Chrome, weil dieser Browser besser als andere Fehler anzeigt. Somit kann ich schneller die Ursache des Problems analysieren.

Abbildung 8: Testprotokol

Hier ist das erstellte Testkonzept. Es wird am Ende der Programmierten Applikation an die Features angepasst und somit werden die wichtigsten Bereiche der Applikation getestet.

2.4 Entscheiden

2.4.1 Komplexe Feature Konzepte Auswerten

In diesem Abschnitt evaluiere ich alle Konzepte, die ich erstellt habe. Ich werde herausfinden aus welchen Gründen welche Konzepte wieso gut sind und welche Konzepte wieso nicht. Anhand dieser Entscheidung werde ich die in meiner Meinung nach besten Lösung für das Problem aussuchen.

|  |  |
| --- | --- |
| **AdminLTE einbinden** | |
| CDN-Konzept | Lokale Dateien Konzept |
| |  |  | | --- | --- | | Positiv | Negativ | | Schnell eingefügt | Viele Links im head | |  | Wenig Übersicht über Links | |  | Braucht eine Internetverbindung | |  |  | | |  |  | | --- | --- | | Positiv | Negativ | | Von AdminLTE Demopage nehmen | Viele files | | Funtioniert ohne Internetverbindung |  | |  |  | |  |  | |
| **Entscheidung**  Der beste Weg AdminLTE einzufügen ist durch die Files in der AdminLTE demo Page. Dort kann ich einfach das Bundle nehmen und dann habe ich Bootstrap, Fontawesome und das AdminLTE. | |

Tabelle 9: AdminLTE einbinden

|  |  |
| --- | --- |
| **Formvalidierung** | |
| Konditional | Validierungsklasse |
| |  |  | | --- | --- | | Positiv | Negativ | | Einfach zu implementieren | Nicht stark OOP orientiert | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | Positiv | Negativ | | OOP-Orientiert | Komplex | | Ich kann anhand eines Arrays die die Formulardaten bearbeiten | Viel Code | | Konfiguration im config file |  | |  |  | |
| **Entscheidung**  Die beste Lösung hier ist die Validierung durch die Validierungsklasse. Diese ist nicht nur OOP orientiert, sondern sie ist auch ausbaubar. In der Zukunft kann ich einfach neue Methoden für die Validierung einfügen. | |

Tabelle 10: Formvalidierung

|  |  |
| --- | --- |
| **Exception Handling** | |
| Try and Catch | Session Exceptions |
| |  |  | | --- | --- | | Positiv | Negativ | | Einfach einzubauen |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | Positiv | Negativ | | Kann für mehr als nur Exceptions verwendet werden | Ist ein wenig schwieriger einzubauen | |  |  | |  |  | |  |  | |
| **Entscheidung**  Ich denke eine Hybrid Methode wäre hier die beste Wahl. Man kann mit Try und Catch die Exception fangen aber sie dann mit der Session anhand eines Keys speichern und dann in der View mit einem Foreach ausgeben, sodass alle Meldungen immer zur View kommen. | |

Tabelle 11: Exception Handling

2.5 Realisieren

2.5.1 Framework Umgebung

Bevor ich alle anderen Features implementieren kann, muss ich sicherstellen, dass alle Framework Teile kommentiert sind und die root .htaccess Datei richtig aufgebaut ist.

2.5.2 AdminLTE Partial Viewkonzept bereitstellen

2.5.3 Persistente Daten

2.5.3.1 DB-Benutzer erstellen

Für das Projekt habe ich einen neuen DB-Benutzer erstellt. Das ist bei Huanga standard.

Table

Description automatically generated

2.5.3.2 Datenbank und Tabellen erstellen

Ich habe anhand der geplanten datenbank die Eigentliche Datenbank erstellt, hier ist diese mit denn relationen aufgeführt. Führ die datentypen habe ich so gut wie möglich evaluiert welche grösse oder datentyp ansich gebraucht wird. Zudem haben einige daten in der Tabelle «client» einen default wert sodass man denn client ausfüllen kann ohne alle daten ausfüllen zu müssen. Dies sollte wichtig sein wenn ein kunde nur schonmal ein kunden konto anlegen will aber noch keine Bankdaten ausfüllen möchte. Bei denn anderen Tabellen sollen alle felder ausgefüllt werden. Hier gibt es keinen ersichtlichen grund nur ein teil der Daten auszufüllen und denn rest nicht.

A screenshot of a computer

Description automatically generated with low confidence

2.5.4 AdminLTE einfügen

Für mein projekt habe ich die Demo page von adminLTE benutzt um ein Layout zu erstellen: «DefaultLayout.php» Hierfür habe alle teile vom AdminLTE Kopiert und diese als das Layout verwendet. <https://adminlte.io/themes/v3/> Die teile die die view ausmachen sind:

* Wrapper «.wrapper» Ein div, das die gesamte Seite umschliesst.
* Hauptheader «.main-header» Enthält das Logo und die Navigationsleiste.
* Hauptseitenleiste «.main-sidebar» Enthält das Seitenleisten-Benutzerfeld, das Suchformular und das Menü.
* Inhalt «.content-wrapper» Enthält die Kopfzeile und den Inhalt der Seite.

Nachher habe ich alle elemente die ich nicht brauche entfernt es gab viele füller elemente die ich nicht brauchte. Um das default layout mit adminlte, bootstrap und fontawesome zu versorgen habe ich von der demo page alle files genommen und in Respektive ordern in meine applikation getan. Dann habe ich das Firmen logo von Caskade eingefügt und mit einer ännlichen schrift denn titel neben dem Logo erstellt.

2.5.5 CRUDs erstellen

Für das erstellen der CRUDs habe ich die einzelenen teil unterschritte alle in die respektiven dateien kategorisiert. Hier ist es wichtig das die erstellung der CRUDs durch sinnvolle erstellung der Dateien gegliedert ist. Das bedeuted ich habe absichtlich mit dem erstellen der entitäten angefangen da ich diese brauche um die Datenbankabfragen in denn Repositories zu machen. Und nachher habe ich erst die Controller erstellt weil ich dafür die Repositories brauche. Und zuguter letzt habe ich die Views erstellt aber das ich nicht wieder zu denn controllers gehen musst habe ich im vorherein mir gedanken gemacht welche views ich brauche und diese Dateien schon erstellt und bennant sodass ich diese namen schon in denn Controllers brauchen konnte. Für mich macht dieser ablauf am meisten sinn weil die gliederung der Dateien erst dann zum zug kommt wann sie gebraucht wird. Auf was ich mich auch noch achten musste war die Seperation of Concerns für die Controllers. Damit hatte ich am anfang probleme und musste eine logische abfolge strukturieren sodass jeder controller richtig organisiert ist und ich nicht einfach die View lade ohne zuerst sie zum beispiel mit daten abzufüllen.

2.5.5.1 Entitäten Klassen erstellen

Hierführ habe ich mit denn Entitäten angefangen. Diese sind sind klassen die alle spalten in der Datenbank als Parameter haben. Zum beispiel habe ich bei der Tabelle User «username» und «password» und diese spiegeln sich wieder als php parameter in der «EntityUser.php» klasse. Zudem hat jede klasse getter und setter sodass ich diese in der View oder im Controller verwenden kann.

2.5.5.2 Repository Klassen erstellen

Nachher habe ich die Klassen erstellt die dafür da sind Datenbank aktionen zu tätigen. In dieser hinsicht hat jede eigene Klasse methoden um Tabellen einträge zu erstellen, löschen, lesen und zu aktualisieren.

2.5.5.3 Controller Klasse erstellen

Die controller habe ich erstellt indem ich mich zuerst an die wichtigsten CRUD aktionen gehalten habe und somit hat am anfang jeder controller änhlich ausgesehen. Nachher habe ich die Controller mit respektiven methoden ergänzt die für diese Repo klassen gebraucht wurden.

2.5.5.4 View Klassen erstellen

Für die Views habe ich verschiedene Konzepte ausprobiert. Wie ich sie mit AdminLTE und Bootstrap designen könnte. Am ende habe ich mich dafür entschieden für die Detail views Tabellen zu benutzen. Hier sollte jede zweite zeile eine andere farbe haben sodass man diese besser identifiezieren kann. Die tabellen sind alle vertikal gegliedert obwohl das doch nicht allzu semantisch ist habe ich mich für diese lösung entschieden weil ich nur so alle spalten anzeigen kann für grosse Tabellen. Für die Edit und Add aktionen habe ich views erstellt die sich an Cards von AdminLTE anlehnen sie sehen ännlich aus wie diese. Für die index Views habe ich inspiration von einem mir bekannten CMS genommen. Contao hat Aktion buttons mit denen man verschiedene Aktionen mit einem Element tätigen kann. Hier für habe ich auch änliche aktion buttons erstellt diese unterscheiden sich farblich und sollten demm benutzer anhand der Icons direkt sagen was passiert wenn man diesen Knopf drückt.

2.5.5.5 User Login Funktionen hinzufügen

User login wurde mit einer Session klasse gebraucht diese setzt die session wenn sie nicht schon gestzt wurde und zudem hat sie getter und setter mit dennen man sessions erstellen kann. Und eine lösch funktion um session zu zerstören. Ich denke das die platzierung der Session klasse im Models ordner sinn macht weil ich dort alle Daten bearbeiten so mache ich das eigentlich auch mit der Session. Zudem musste ich die Wichtigsten login funktionen einfügen dafür habe ich neue Methoden erstellt die respektiv denn benutzer einloggen, verfizieren und Ausloggen. Das generelle Konzept der Passwort reset funktion habe ich mir inspiration von einigen echten Applikationen genommen. Diese haben alle ein passwort reset feature. Dieses erstellt ein Token der einzigartig ist und nur einmal benutzt werden kann um ein Passwort zurück zusetzten. Ich habe das gleiche prinzip für meine Applikation verwendet. Beim eingeben des Benutzernamens das Zurück gestzt werden. Wird ein Token erstellt der eine Relation zum benutzer hat der das Passwort zurück setzen wollte. Da der Token auch in der URL als args verwendet werden kann. Könnte man dieses Feature in der Zukunft mit einem SMTP server erweitern und dann dem benutzer eine email schicken und von der Kann er dann das Passwort zurück setzen. Dies werde ich nach der IPA als Feature erstellen.

2.5.6 Formular Validiedrungskonzept einbauen

Das Valiederungskonzept das ich erarbeitet habe besteht aus drei teilen. Ich habe eine Validation klasse erstellt hier werden alle Check methoden abgearbeitet in einer zentralen Validate methode. Diese check methoden überprüfen ob das Validation array richtig mit POST daten ausgefüllt worden ist. In der Regel ist diese Valiederung als Serverside Validation bekannt. Aber die meisten Use cases von anfang an abzufangen habe ich HTML5 strikte attribute die sicherstellen das dem beutzer gezeigt wird was von ihm verlangt wird beim Ausfüllen des Formulars. Der letzt teil der Validierung der daten findet im respektiven Repo der Controller aktion statt. Hier habe ich ein array erstellt das die aufgebauten HTML5 attribute valididert anhand der definierten Parameter.

2.5.7 Exception Handling Konzept einbauen

Das exception handling habe ich mit Try and catch eingebunden. Hierfür versuche ich denn try block abzuarbeiten und wenn das nicht funktioniert, in catch eine neue exception zu erstellen und diese dem benutzer in der View zu zeigen. Es gibt hier ein paar spezielle varianten einige methoden können mehr oder weniger Exception handling funktionalität haben als andere weil einige methoden einfach komplexer aufgebaut sind. Ich musste aus zeit gründen ein wenig von meinem orginalen Konzept abwewichen und habe mich hier für eine einfacher implementation entschieden.

2.5.8 Statistiken einfügen

Die Statistiken habe ich mit zwei neuen Views und einem neuen Controler gelöst. Hierfür habe ich in existierenden Repos datenbank abfragen erstellt. Im controller werden diese nur weitergeleitet und instantziert. Für die View habe ich spezielle dafür foreach benutzt und im controller habe ich Parameter definiert um um die anzahl jahre und Monate in der view anzuzeigen.

2.5.9 Autocomplete

Das Autocomplete feature war nicht von anfang an geplant es zu implementieren. Da ich aber im nachhinein festellen musste das die Applikation ohne nicht funktionieren kann. Ich habe das autocomplete feature so eingebunden das, bei einer suche anhand von Javascript das ein Json als antowrt gibt und das wird von Jquery-UI abgearbeitet. Für die implementation vom löschen eines eintrags habe ich key up verwendet das schaut wie gross der String im input field ist und wenn er leer ist die Korrespondierende ID input field hidden löscht sodass dieses nicht im POST auftaucht.

2.6 Kontrollieren

Die Kontrolle der Applikation findet mit in Plannung erstellten Test Konzept statt. Hierführ verwende ich das erwähnte konzept und Reflektiere über die Features und welche evtl. noch die applikation noch ergänzen würden. Durch die kontrolle ist das einfacher da ich mich auf die funktionen der Applikation konzentriere, kann ich schneller evaluieren was noch fehlt oder welche funktionalität in der Zukunft hinzugefügt werden soll.

2.6.1 Testprotokol

Alle Testprotokole wurden von Aaron Schreiber am 08.05.2023 geschrieben.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Feature / Zeitpunkt des Tests | Nr. | Testfall | Vorgehen | Zu erwartendes Ergebnis | Resultat | Massnahmen / Empfehlungen | Bemerkungen |
| CRUD Aktionen /  15:00 uhr | 1 | Alle CRUD-Aktionen und nachher schauen, ob diese in der Datenbank die gewünschten Resultate geben. | Jedes CRUD wird ausgefüllt und abgeschickt nachher wird in der Datenbank geschaut ob das Resultat denn Benutzereingaben ensprechen. | Die datenbank sollte anhand der $\_POST daten richtig die daten eingestzen, updaten oder Löschen. | Jedes CRUD funktioniert aber ein feld: «Geburtstag» Sollte überpüfen ob der Kunde älter als 18 jahre alt ist. | Höchst warscheinlich werde ich es zeitlich nicht mehr während der IPA schaffen dies zu beheben werde das aber Nach der IPA ASAP vornehmen. | Eine kleinere Anpassung die aber im Kontext der Applikation wichtig ist. |
| Routing /  15:40 uhr | 2 | Das Routing sollte den richtigen Controller und die richtige Aktion nachweisen. | Alle Controller und Aktionen werden überprüft und ob sie in der URL das gewünschte Resultat liefern. | Die URL spiegelt im MVC Framework das Routing wieder und sollte anhand von dem den Jeweils richtigen Controller und Aktion anzeigen. | Alle Controller und Actions zeigen die Respektiven controllers und actions an. Alles funktioniert wie erwartet. | - | - |
| Navigation Path /  16:00 uhr | 3 | Die Navigation sollte zur richtigen Seite weiterleiten. | Die navigation der ganzen appliaktion und das Testen der Umleitungen. Werden anhand aller Controllers und actions getestet zudem werden redirects vom Login und Passwort reset getestet. | Die weiterleitung von einer Page auf die nächste anhand von Richtigen eingaben. Oder falsche eingaben sollte zur selben seite weitergeleiteten. Natürlich hier wichtig ist die Respektive seite und Controller und Aktion. | Alle controllers und actions verweisen auf die Respektive seite. Die redirects der Login seite und der User controllers und actions funktioniert auch. | - | - |
| Index Aktion /  16:30 uhr | 4 | Der Startpunkt der Applikation sollte beim ersten Mal angezeigt werden. | Die Applikation soll mit dem einloggen getestet werden und hier soll geschaut werden wohin verwiesen wird. Anders kann man die index seite nicht testen weil wenn man nicht eingeloggt ist wird man zur login seite Weitergeleitet. | Die applikation sollte immer einen festen startpunkt im Login haben und wenn man eingeloggt ist auf der Vertrag index seite. | Die applikation verweisst nach dem Login auf die Client index action. Das ist falsch. | Ich habe in der Login action vom user controller die weiterleitung angepasst und sie verweisst jetzt an denn richtigen ort. Jetzt wird das gezielte ergebniss ausgeführt. | Der Test aht ergeben das die Weiterleitung nicht stimmte ich musste diese anpassen da das nicht viel zeit in Anspruch nimmt habe ich das schnell erledigt. |
| Session /  16:50 uhr | 5 | Die Login Seite sollte ohne Session respektiv nicht gesetzte Session Schlüssel nicht angezeigt werden. | Alle möglichen controllers und actions werden in der URL eingeben und alle sollte wenn man nicht eingeloggt ist zur login page weiterleiten. | Wenn kein session mit dem Key Benutzer (in der Applikation auf englisch: „user key has to be set“) existiert dann sollte die Applikation zurück zum Login weiterleiten. Ohne session key benutzer können keine anderen seiten in der Applikation geöffnet werden. | Funktioniert kein Problem aufgetaucht. | - | - |
| Valiedierung / 17: 20 uhr | 6 | Die Validierung sollte anhand der angegeben Validation array die Daten richtig überprüfen. | Die validierung aller Felder von allen CRUDs durch die ganze applikation hinweg sollte anhand der Validation array eigenschaften die gewünschten ausgaben anzeigen oder fehlermeldungen. | Hierführ entferne ich die HTML5 attribute im browser und überprüfe was die Fehlermeldung ausgeben und ob diese angepasst werden sollten oder ob die daten in der DB richtig aufgebaut sind. | Alle Validierungen funktionieren. | - | - |

2.6.2 Test der Applikation Fazit

Meine applikation hat alle der Aufgabenstllung gewünschten funktionen. Dennoch sehe ich das nach dem Kontrollieren und Konzentriertem testen der Applikation einige features unbefriedigen integriert sind oder noch ausgebaut werden sollten.

Eines dieser Features sind die Breadcrumps für die Seiten navigation. Diese existiert nur anhand eines fixen strings das sollte in der Zukunft nach der IPA verbessert und Dynamisch erstellt werden.

Ein weiteres Feature sind benutzer ausgaben in der View. Meines empfindens nach sollte der Benutzer darüber informiert werden wenn eine insert in die Datenbank erfolgreich war. Die applikation verfügt über exception handling aber sie sollte noch eine weichere beutzer ausgaben handling feaature haben. Ich werde dieses Feature nach der IPA auch noch einbauen.

Das letzte feature das Meines erachtens wichtig ist, ist die pagination diese existiert derzeit nicht und sollte definitiv Post IPA eingebaut werden. Ohne diese feature werden grosse datenmendgen mühsam anzuschauen in der Applikation.

2.7 Auswerten

2.7.1 Selbstreflexion und Fazit

3 Anhang

3.1 Quellenverzeichnis

|  |  |  |  |
| --- | --- | --- | --- |
| Medium | Quelle | Infos wurden in welchem Kapitel verwendet | Entnommene Informationen |
| Webseite | <https://www.phinst.unibe.ch/ueber_uns/lehrlingsausbildung/website_polymechaniker/ausbildung/iperka/index_ger.html> | Ganzes IPERKA Kapitel | Allgemeiner IPERKA Aufbau |

3.2 Abbildungsverzichnis

Abbildung 1: IPERKA Ablauf Seite 9

Abbildung 2: Dokumentenstrukturierung Seite 10

Abbildung 3: Zeitplan IPA-Bericht Seite 12

Abbildung 4: ERM Seite 18

Abbildung 5: AdminLTE Seite 19

Abbildung 6: AdminLTE 3 Cards Seite 19

Abbildung 7: Beispiel Tabelle Seite 19

Abbildung 8: Testprotokol Seite 20

3.3 Tabellenverzeichnis

Tabelle 1: Dokument Information Seite 3

Tabelle 2: Hardware Seite 6

Tabelle 3: Software Seite 7

Tabelle 4: Personen und Rollen Seite 8

Tabelle 5: Termine Seite 8

Tabelle 6: Durchführungstermine Seite 8

Tabelle 7: Versionierung IPA-Bericht Seite 11

Tabelle 8: Meilensteine Seite 17

Tabelle 9: AdminLTE einbinden Seite 21

Tabelle 10: Formvalidierung Seite 21

Tabelle 11: Exception Handling Seite 22

3.4 Abkürzungsverzeichnis

3.5 Glossar

3.6 Quellcode