Pflichtenheft

Registrierung und Maschinenpark - Kundenportal / MyPÖTTINGER

Änderungen

VERSION	DATUM	ÄNDERUNG	BETEILIGT
0.1	13.05.2020	Grundstruktur des Dokuments	PÖ:AuguDom
0.2	14.05.2020	Ausformulierung	PÖ:AuguDom HTL HoerVal HTL SchoAle

Inhalt

1. Zielbestimmung

Ziel ist die Erstellung eines Registrierungsformulars sowie einer Maschinenverwaltung für die Endkundenplattform für Pöttinger-Kunden.

Die Module sollen dem aktuellen Stand der Technik entsprechen, aufgrund der erwarteten Massendaten insbesondere im Datenhaltungsbereich skalierbar und soweit modular sein, dass einzelne Teile getauscht werden können ohne übermäßigen Adaptionsaufwand in den jeweils anderen Schichten zu erzeugen.

Eine Erweiterbarkeit der für den Endkunden sichtbaren Funktionen ist ein Muss-Bestandteil und eine solche Erweiterung, insbesondere Fehler einer solchen, dürfen keine Auswirkung auf die anderen angebotenen Funktionen haben.

Es müssen mit einem User-Interface sämtliche Kunden-Geräte (Desktop, Tablet, Smartphone) bedient werden.

Weiters muss ein RBAC[i] implementiert sein, welches erlaubt den Nutzern aufgrund ihrer Rollen verschiedene Module zugänglich zu machen.

Wo möglich und ausreichend, sollen Standardkomponenten der jeweiligen Frameworks verwendet werden. (Benutzer-/Rechtemanagement, Internationalisierung, Schnittstellen-Libraries, Sicherheit, Templates, …)

2. Produkteinsatz

a. Anwendungsbereiche / Zielgruppen

Die Anwender sind Pöttinger-Kunden, welche über die gesamte Welt verstreut sind. Das Userinterface sollte somit, für einen typischen Smartphone-Nutzer, ohne weiteren Schulungsaufwand benutzbar sein. Einerseits sollen gängige UX/UI Standards beachtet werden, andererseits Overlay- bzw. Inline-Informationen bei Fehlern oder komplexen Formularen.

b. Betriebsbedingungen

Es wird erwartet, dass ein Großteil der Aufrufe über Smartphones geschieht, daher ist sowohl auf Größe der Anwendung wie auch auf die clientseitige Logik auf dieses Szenario hin zu optimieren.

3. Produktübersicht

a. Produktfunktionen / Projektumsetzung

Nutzer können sich selbst registrieren und anmelden. Der Nutzer kann seinem Konto selbstständig Pöttinger Maschinen hinzufügen. Mit den eingepflegten Maschinen bekommt er einen direkten Zugriff auf weitere Daten (Betriebsanleitung, Wartungspläne, Ersatzteile, …). Sollte die Maschine auch eine EDV-Schnittstelle besitzen, werden Auswertungen zu den Einsätzen präsentiert und Hinweise zu fälligen Wartungsaktionen ausgegeben.

Ausblick:

Diese Information sollte in einem späteren Release auch als Push-Nachricht an das dann “PWA-Enabled” Angular Frontend möglich sein, bis dahin zumindest (evtl. konfigurierbar) per E-Mail, oder SMS. In der Benachrichtigung ist wiederum ein Link enthalten, über den die konkret fällige Wartung angezeigt wird.

Es gibt grundsätzlich zwei Benutzergruppen: Private / Einzelpersonen, und Betriebe. Bei Betrieben sollen bereits während der Anmeldung weitere Daten in Erfahrung gebracht werden.

b. Anforderungen an das Registrierungsformular:

b.1. Grundanforderungen

State-Management der Formularfelder & Gruppen (Gesendet, Geprüft, Fehlerhaft, …), um die Last & Kosten für Systeme und Pay-per-Use Services minimal zu halten.

Mehrgliedriges Formular (Multistep), aufgebaut aus verschiedenen Components, die auch beim Bearbeiten der Daten nach Abschluss der Registrierung wieder zum Einsatz kommen. Die Adress-Gruppe kommt bei Firmenkunden zweimal im Registrierungsformular zum Einsatz, zudem sollen die Daten nach der Registrierung auch über eine Profilseite wieder geändert werden können.

Die Definition der Felder muss einfach zu ändern / erweitern sein, die Darstellung im Front-End muss dynamisch, aufgrund einer Struktur, zur Laufzeit erfolgen.

b.2. Security

Absicherung gegen gängige Webattacken (Insbesondere durch CSRF-Token + Validierung im Backend! und Google Captcha V3). Verwenden eines Token (JWT) zur Benutzeridentifikation nach Anmeldung.

b.3. Internationalisierung

Unabhängige Auswahl von Sprache und Land. Anpassung der Reihenfolge (z.B.: PLZ vor / nach Ort), Bundesstaat (z.B.: ja/nein, Eingrenzung), und Validierung statischer Länderparameter (z.B.: Länge und Beschaffenheit von Postleitzahlen) unter Einbeziehung bestehender Datenquellen.

Recherche nach bestehenden Datenquellen für länderspezifische Anschriften, Telefonnummern Formatierungen / Vorwahlen und Auswahl einer geeigneten. Erstellen einer Struktur welche im Frontend und Backend die Spezifika enthält und damit die Darstellung und Validierung steuert.

Verschiedene Encodings (China, Russland, …) müssen korrekt gespeichert und auch wieder dargestellt werden.

b.4. UI / UX

Unterstützung des Nutzers bei Adresseingabe:

- Über Browser/Geräte-Standort und Reverse-Lookup
- Über z.B.: Google Maps API Autocomplete
- Über ein freies Formular (welches auch im Anschluss an die ersten beiden Varianten vorausgefüllt verwendet wird jedoch weitere Korrekturen zulässt)

Validierung je Eingabegruppe und klare Fehlermeldungen direkt bei den fehlerhaften Eingaben. Farbliche Unterscheidung zwischen fehlenden (bei Pflichtfeldern) passenden und falschen Eingaben.

b.5. Umsetzung

Sämtliche externe Dienste, sollen so weit möglich, über das .net Core Backend angebunden sein, um dort gegebenenfalls rate-limits und weitere Maßnahmen gegen missbräuchliche Nutzung treffen zu können, sowie die API-Keys nicht über das Angular Kompilat ausliefern zu müssen.

Im Backend müssen sämtliche Eingaben nochmals validiert und normalisiert werden, um gegen einen Nutzereingriff im Browser oder der HTTP Kommunikation geschützt zu sein.

b.6. Überprüfung der Telefonnummer

Erstellen / Einbinden eines Dienstes welcher per Anruf oder SMS den Halter der Telefonnummer bestätigt.

b.7. Tests

Tests des Backends müssen direkt über Swagger im Browser möglich sein, zudem muss die essentielle Funktion mit Unit-Tests abgesichert sein. Die Tests sollen bei Zusammenführung in der GIT-CI Pipeline sicherstellen dass keine Releases veröffentlicht werden, bei denen die Anmeldung nicht möglich ist.

Tests der elementaren Funktion des Frontends müssen automatisiert möglich sein.

b.8. Erweiterungen

Wenn sämtliche Anforderungen aus dem Registrierungsprozess erfüllt sind:

Erstellung oder Anbindung an ein Dashboard zur Darstellung der Nutzerbasis (insb. geographische Gruppierungen und Charts zur Visualisierung von Neu-Konten über einen zeitlichen Horizont).

Erweiterung der MyPÖTTINGER Plattform um ein weiteres Modul, wie z.B.: Erstellung eines Maschinenparks (Speichern von PÖTTINGER Maschinen in dem Nutzeraccount und anzeigen weiterer Informationen zu diesen).

b.9. Recherchen / Präsentationen

Für jeden externen Service (Reverse-Geo-Lookup, Telefonnummern-Validierung, Nationale Adressstruktur Quellen, Type-Ahead Adresssuche, Dynamische Validierung, Bot-Protection, …):

- Evaluierung des Marktes (Alternativen)
- Schwächen Stärken + Abgleich mit Anforderungen aus dem Projekt
    - Statische Lösung möglich / Existent?
    - Regionale Verfügbarkeit
    - Kosten
    - Synergien mit weiteren Anforderungen
    - …

- Dokumentation und Präsentation
- Implementierung

c. Produktdaten

Im ersten Release: Benutzer, Rechte, Maschinen, Maschinendaten, Wartungen, Maschinendokumente, Ersatzteile.

d. Produktleistungen

Die Plattform muss mit den erwarteten Massen an Maschinendaten umgehen können, zudem mit den aktiven Nutzern skalieren, um nicht immer die gesamte Rechenleistung vorhalten zu müssen.

e. Qualitätsanforderungen

Merkmal \ Stufe	Sehr gut	Gut	Normal	Nicht relevant
Funktionalität			x	
Zuverlässigkeit			x	
Benutzbarkeit	x			
Effizienz		x		
Änderbarkeit	x			
Übertragbarkeit			x	

Ausführung:

- Keine Personen / Finanzkritischen Operationen,
- Keine hochkomplexen Algorithmen
- Muss selbsterklärend sein
- Muss mit Massendaten / Nutzern umgehen
- Wird auf lange Zeit eingesetzt, Änderungen bereits im Vorfeld sichtbar
- Übertragung ist nicht angedacht, kontinuierlicher Ausbau der Funktionen und Benefits

f. Benutzungsoberfläche

Moderne Single-Page-Application, Mehrsprachigkeit, Responsive-Design.

g. Nichtfunktionale Anforderungen

Aktionen sollten nachvollziehbar und dokumentiert sein. Logging von Login/-out, Update / Create / Delete, erstere insbesondere bei fehlgeschlagenen Versuchen.

Die Übertragung sämtlicher Daten erfolgt verschlüsselt entsprechend Industriestandards.

Monitoring der Performance und Fehler bis in die Client-Ebene.

Tests der zentralen Backend Funktionen, als Unit Test und/oder via PostMan.

Die Plattform soll den gängigen Angriffen im Web keinerlei Ansatzpunkte liefern. Insbesondere darf kein Input die Funktion einer Abfrage verändern (Schutz z.B. durch Einsatz eines OR-Mappers oder Prepared Statements), Replay/Cross-Site-Request-Forgeries (CSRF- / One-

Time-Tokens), Brute-Force-Versuchen, Eingaben müssen normalisiert und ohne HTML Sonderzeichen abgelegt werden, weitere siehe OWASP Top 10-2017[1].

Sämtliche Gesetze und Verordnungen werden sowohl von der Plattform wie auch genützten Services eingehalten bzw. die Durchsetzung derer unterstützt (insbesondere die DSGVO). Eingebundene Services sind auf deren DSGVO Tauglichkeit zu prüfen und entsprechende Textvorlagen als Information für die Endnutzer bereitzustellen.

4. Technische Produktumgebung

a. Software

Im Backend kommt .net Core als zum Einsatz, das Frontend ist in Angular 9+ entwickelt.

b. Hardware / Infrastruktur

Aufgrund der erwarteten Lastschwankungen und des sowohl tiefen wie breiten Monitorings der Software-Anwendung wurde Azure Cloud als Ziel-Plattform identifiziert.

Ein Datenaustausch wird mit Pöttingers SAP ERP System und im Laufe des Projekts zu identifizierenden weiteren Systemen notwendig. Weitere Übergabe-Punkte finden sich in Punkt d. Produktschnittstellen.

c. Organisation

Das Environment muss so gestaltet sein, dass der parallele Betrieb von Test- und Produktivsystemen frei von Seiteneffekten möglich ist.

Es werden mehrere Entwickler gleichzeitig an dem Projekt arbeiten, eine Nachvollziehbarkeit der Änderungen ist notwendig, zudem soll diese über die erledigten Issues / Milestones einen aktuellen Status über den Fortschritt der Entwicklungen geben.

Das Projekt ist in eine CI / CD Umgebung integriert und erstellt bei Zusammenführen in einen Master-Branch neue Releases, welche wenig später in der Testumgebung gehostet sind. Zudem kann über eine lokale Docker Umgebung Änderungen live in einen Browser am gleichen Rechner

d. Produktschnittstellen

Im Zuge der Beratung soll ein Protokoll zum Austausch der Daten zwischen Frontend und Backend der Plattform ausgewählt werden.

Weitere Schnittstellen, über welche Produktdaten, Kunden… , mit dem Pöttinger ERP & CRM Systemen abgeglichen werden sind in Planung. Eine Entscheidung in welcher Art dieser Abgleich zwischen den Systemen, etwa real-time, täglich, … erfolgt zu einem späteren Zeitpunkt.

Auch die Frontend-Anwendung sollte möglichst entkoppelte Komponenten aufweisen, welche gemeinsam auf gewisse Services zugreifen können. Aufgrund der gewünschten kompakten Gesamtgröße sollten dies nicht verschiedene SPAs sein, sondern idealerweise Module.

Ansatz: Wenn vom Umsetzungsaufwand noch vertretbar, Angular Modules, eingebunden als Libraries, bzw. während der Laufzeit als UMD nachgeladen. Sollten zum Projekt-Start bereits

[1] https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

stabile Ivy Builds in Aussicht sein, muss der Aufwand aufgrund der reduzierten Bundle-Größe neu bewertet werden.[2] [3]

Nutzer sollen ihr Konto auch mit dem AgriRouter koppeln können. Dieser bietet Zugriff auf Maschinen-Stammdaten welche sich der Nutzer bereits in anderen angebundenen Lösungen erstellt hat. Diese sollen in myPöttinger übernommen werden. Der Zugriff auf diese Daten wird z.B. von NextFarming unter https://github.com/ManuelB/agrirouter-api-protobuf-definitions dokumentiert.

Für einen externen IoT Gateway muss eine Schnittstelle angeboten werden. Über diese werden Einsatzdaten der Landtechnik-Maschinen in das Endkundenportal eingespielt.

Als interne Schnittstelle zwischen den Komponenten ist besonders auf die Authentifizierung und das Session-Management zu achten.

5. Spezielle Anforderungen an die Entwicklungsumgebung

Die Entwicklungsumgebung soll möglichst alle verwendeten Komponenten unterstützen. Dies wären einerseits die angepeilten Sprachen und Frameworks, andererseits auch die Versionsverwaltung der Dateien sowie das Überspielen / Monitoren des Projekts in Azure.

Falls aufgrund der gewählten Schnittstellen zwischen den Plattformen notwendig, sollte auch die Dokumentation der Aufrufe automatisch aus der Definition generiert werden. (Swagger falls man sich für eine REST basierte Variante entscheidet, OData/SOAP beschreibt sich selbst).

6. Gliederung in Teilprodukte

a. Frontend

Angular, ausgeliefert von Azure. Schnittstellen zum Backend, Google Captcha, CDNs, evtl. Authenticator.

b. Backend

.Net Core, ausgeführt in Azure. Schnittstellen zu Frontend, Pöttinger ERP und externen Services.

c. Pöttinger Services

Über SOAP / OData erreichbar.

d. Externe Services

Sicher: AgriRouter, Google Captcha, CDN's.

Je nach Beratung: Authenticator (Auth0), + weitere.

[2] https://github.com/angular/angular-cli/wiki/stories-create-library
[3] https://github.com/lmeijdam/angular-umd-dynamic-example

7. Ergänzungen

a. Erwartungen an die Beratungsleistung

Funktionsfähige Applikation bei der sämtliche Kern-Bestandteile vorhanden sind, sodass eigenverantwortlich weiterentwickelt werden kann.

Entscheidungshilfe für die Schnittstelle und die Häufigkeit (On-Demand vs. zeitlich getriggert) des Abgleichs der Daten zwischen den Backend Systemen (Kundenportal und Pöttinger SAP ERP).

Bieten einer Entscheidungsgrundlage für die Platzierung der Authentifizierungslösung, z.B.: einen Offsite-Authenticator wie Auth0, eine im .net Core Framework enthaltene Variante wie ASP.net Core Identity oder Eigenentwicklung, inkl. Umsetzung zumindest eines Berechtigungschecks gegenüber der gewählten Lösung. Ebenso soll hierbei entschieden werden wo die Session-Daten verwaltet werden und wie diese übermittelt werden (Cookie, Redis, JWT, …)

Beratung welche Ansätze aus Microsofts Vorzeige Applikation eShopOnWeb[4] und dem dazugehörigen eBook[5], AKS, oder .net Frameworks wie Steeltoe für das gegenständliche Projekt sinnvoll sind, bzw. allgemein wie weit man sich von einem IaaS zu einem CaaS-Setup bzw. sogar in Richtung FaaS wagen soll.
Zusätzlich wäre dann zu klären ob die 12Factor App Principles auch in der .net / Azure Welt in vollem Umfang sinnvoll sind.

Insbesondere die Cloud-Architektur ist soweit konfiguriert, dass man damit einen Go-Live vornehmen könnte.

Information über die Realisierung der Datenübertragbarkeit in anderen Projekten. Pro-Aktiv einen „Meine-Daten“ Export anbieten?

Die grundlegenden Schritte um eine Arbeitsumgebung (Visual Studio) so zu konfigurieren, dass damit das Projekt bearbeitet werden kann, sind, ebenso wie die Einstellungen des Cloud-Services, inkl. Begründung zu dokumentieren.

Fortlaufende Beratung und Information über neue Schwachstellen in den verwendeten Komponenten und die Behebung dieser, etwa über Aufnahme in einen E-Mail Verteiler.

8. Glossar

a. Abstraktionsschichten Cloud-Anwendungen:

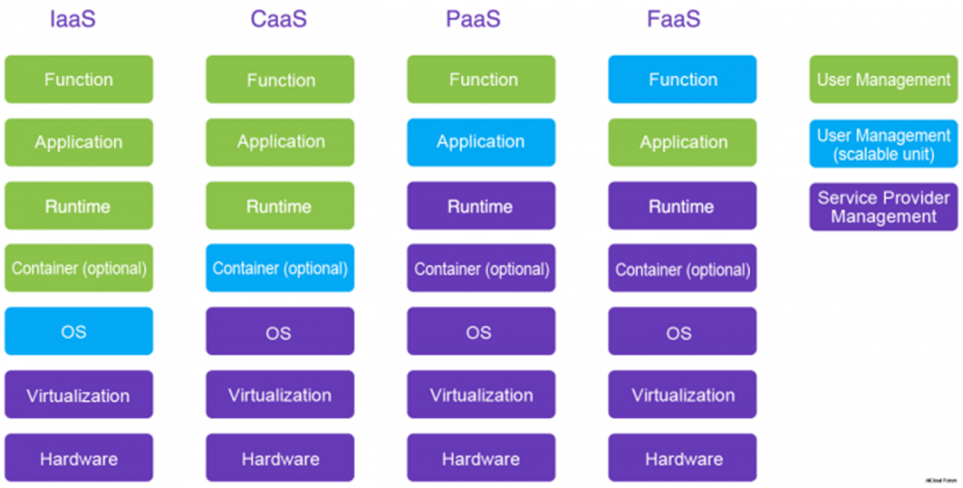
IaaS: Infrastructure as a Service - “Virtual Machine”

CaaS: Container as a Service - Bereitgestellte Container Verwaltung / Docker & Kubernetes

PaaS: Platform as a Service – Entwicklungsumgebung / Deployment, ... liegt ebenso beim Cloud-Anbieter

[4] https://github.com/dotnet-architecture/eShopOnWeb
[5] https://aka.ms/webappebook

FaaS: Function as a Service – nur noch die Geschäftslogik als Funktion wird implementiert, sämtliche Aufrufe sind zustandslos.



b. Kürzel

AKS: Azure Kubernetes Service

OR-Mapper: Objektrelationale Abbildung (Instanz einer Klasse wird aus Entität einer DB erzeugt)

[i] RoleBasedAccessControl