So gelingt die Entwicklung einer Management-Plattform für Fahrgastinfos

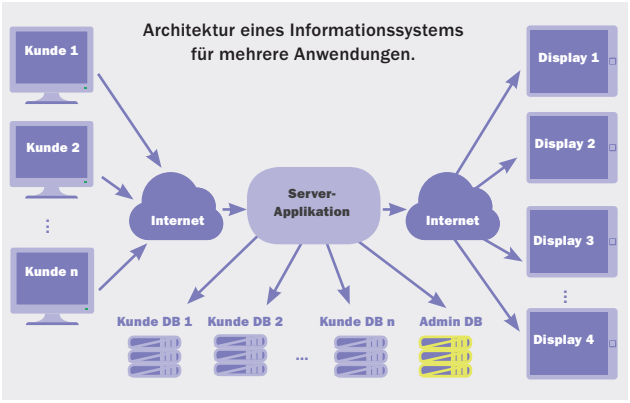
Der Einsatz von Continuous Integration und Continuous Testing gepaart mit früher Einbeziehung des Kunden führt zu effizienter Softwareentwicklung, auch wenn sich die Anforderungen während der Umsetzung ändern. Es gilt allerdings, auch technische Aspekte zu berücksichtigen.

Bei Fahrgastinformationen von Verkehrsbetrieben und anderen digitalen Informationsstellen bedarf es einer entsprechenden Infrastruktur für die Verwaltung der Displays und deren Inhalte. Darüber hinaus müssen auch Parameter wie Ladebestände, Temperatur und der «Gesundheitszustand» der Displays überwacht werden, um rechtzeitig auf Unregelmässigkeiten reagieren zu können. Da die Displays an verschiedenen Orten installiert und über Mobilfunk vernetzt werden, ist eine effiziente Bandbreitenutzung besonders wichtig.

Konsequent mit DevOps

Bei der Entwicklung der Lösung sind effiziente DevOps-Prozesse essenziell, die sich auf eine gute Test-Abdeckung auf allen Ebenen stützen. Ausser den Unit-Tests der Entwickler gehören dazu auch automatisierte Integrations- und Serviceschnittstellentests. Denn ein hoher Automatisierungsgrad ermöglicht eine schnelle Nachprüfung durch Regressionstests und verkürzt die Zeiten zwischen Entwicklung und Auslieferung. Manuelle GUI-Tests – idealerweise vom Kunden durchgeführt – tragen ebenfalls zur Qualität der Software bei und stärken das Vertrauen des Kunden. Einfache Smoke-Tests wiederum ermöglichen eine schnelle Überprüfung in allen Umgebungen und schützen vor unangenehmen Überraschungen.

Für eine ungestörte Arbeit aller Parteien empfehlen sich mehrere Umgebungen. Weiter sollte jeder Entwickler eine eigene Umgebung haben. Bei den Unit-Tests genügt es, die Schnittstellen der Displays mit einfachen Mocks zu simulieren. Die Staging-Umgebung hingegen entspricht der produktiven Umgebung inklusive der effektiven Displays und dient dem automatisierten Testen sowie der Durchführung manueller GUI-Tests durch den Kunden. Dieses Set-up erlaubt es, die Ausfallzeiten der produktiven Umgebung bei Software-Updates auf ein Minimum zu beschränken. Es genügt nämlich, nach der Installation einer neuen Version ihre Lauffähigkeit durch Smoke-Tests zu verifizieren.



Icons: macrovector_official / Freepik

Der Autor
Jiri Petr, Bereichsleiter Applications, CSA Engineering

Technische Aspekte

Neben der Zusammenarbeit zwischen den Softwareentwicklern und dem Kunden gilt es auch, technische Aspekte zu berücksichtigen. Dabei kann auf eine Microservice-Architektur verzichtet werden. Die Serverapplikation liegt also als eine einzige Applikation vor. Diese Lösung vereinfacht vor allem die Inbetriebnahme und den Betrieb wesentlich. Da solche Applikationen üblicherweise für mehrere Kunden eingesetzt werden, ist Mandantenfähigkeit Voraussetzung. Die Kombination von separaten Datenbanken pro Kunde mit einem zentralen Verzeichnis der Konfigurationsdaten ermöglicht eine schnelle Aufschaltung neuer Kunden. Auch gilt es, die Wahl der Protokolle für die Ansteuerung der Displays zu berücksichtigen. So sind etwa REST-Services mit JSON weniger effizientere Technologien als etwa MQTT oder gRPC. Ein Ersatz von JSON durch binäre Formate wie Protobuf wirkt sich erfahrungsgemäss ebenfalls positiv auf die Bandbreitennutzung aus.

Fazit und Learnings

Continuous Integration, Continuous Testing und Continous Deployment verkürzt die Entwicklungszyklen und erlaubt eine schnellere Inbetriebnahme der Software beim Kunden. Die einzelnen Komponenten der Microsoft Azure DevOps Tool Chain sind gut aufeinander abgestimmt, was im Vergleich mit der Open-Source-Welt das Aufsetzen der benötigten Build- und Testumgebungen vereinfacht. Überdies stärkt eine frühe Einbeziehung des Product Owners beim Kunden in die Tests und in die konzeptionellen Entscheide das Vertrauen zum Lieferanten und zum Lieferobjekt. Dazu ist es von Vorteil, wenn der Product Owner über ein gutes technisches Verständnis verfügt.

Den Beitrag finden Sie auch online
www.cetoday.ch