Systemarchitektur

In diesem Kapitel wird exemplarisch die im Rahmen des NuR.E-Projekts entwickelte Systemarchitektur beschrieben. Je nach Anforderungen an das System, kann sich jedoch auch durchaus eine andere Systemarchitektur besser eignen. Die einzelnen Bausteine der in diesem Projekt umgesetzten Systemarchitektur und die Zusammenhänge zwischen diesen, sind in Abbildung 15 schematisch dargestellt.

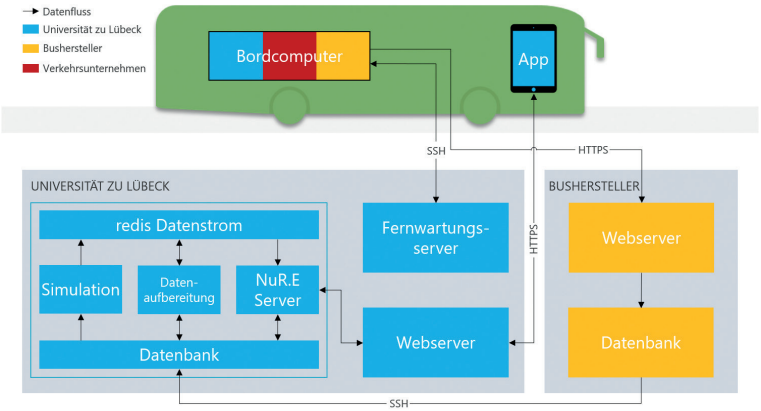

Abbildung 15. NuR.E Systemarchitektur

Nicht alle dargestellten Komponenten sind für eine korrekte Funktionsweise des Demonstratorsystems notwendig. Der Fernwartungsserver ist speziell auf den Bordcomputer abgestimmt und dient nur zur Fehleranalyse bei Problemen mit der Datenübertragung zwischen dem Bus und

der Datenbank des Busherstellers. Aus Sicherheitsgründen ist diese Funktionalität vom Rest des NuR.E-System abgekapselt.

Auch die Daten-Simulation ist keine essentielle Komponente des Systems. Sie unterstützt ausschließlich die Entwicklung des Systems, indem beispielsweise bestimmte Szenarien oder Fahrmanöver gezielt getestet werden können. Dies hat sich während der Entwicklung als enorm hilfreich erwiesen.

In den folgenden Unterkapiteln werden einzelne Komponenten der Systems näher beschrieben und wesentliche Konzepte erklärt. Zudem werden mögliche alternative Herangehensweisen kurz diskutiert.

Anzeigetablet

Als Anzeigegerät wurde im NuR.E Projekt ein iPad Mini gewählt. Das Tablet erfüllt alle oben beschriebenen Anforderungen an Displaygröße, Pixeldichte, Helligkeit und Kontrast. Weiterhin bietet Apple mit dem Supervised Mode einen sehr umfangreichen Kiosk-Modus: Schaltet man diesen Modus über die Apple Entwickler- und Management-Tools frei, stehen umfangreiche administrative Funktionen zur Verfügung. Der Single App Mode erlaubt es beispielsweise, eine App auszuwählen, die permanent geöffnet bleibt. Der Benutzer hat in diesem Modus keinerlei Möglichkeiten, die zuvor festgelegte App zu schließen. Zudem bietet der Single App Mode den Vorteil, dass auch nach einem Neustart des Gerätes sofort wieder die zuvor festgelegte App automatisch geöffnet wird. Der Single App Mode funktioniert nur mit Standalone-Applikationen. Es gibt jedoch eine ganze Reihe an Apps, die in einem integrierten Browser eine zuvor festgelegte Website anzeigen können. Eine solche App in Kombination mit dem Single App Mode bietet so die vielversprechendste Lösung. Als problematisch erwiesen sich jedoch die Updates von Apple, die regelmäßig zu Problemen mit dem Single App Mode geführt haben. Daher sollten automatische Updates auf dem Gerät deaktiviert werden.

Auf Android sind derartige Lösungen (zum Zeitpunkt dieses Projektes) leider nicht, beziehungsweise nur in einer eingeschränkten Form, verfügbar.

Im Praxistest hat sich ein iPad Mini im Single App Mode in Kombination mit der App Kiosk Pro Plus als eine für das NuR.E-System geeignete Grundlage erwiesen. Diese bietet umfangreiche Konfigurationsmöglichkeiten und auch eine JavaScript-API, mit der auch der Zugriff auf verschiedene Geräte-Funktionalitäten möglich ist. So können über die API beispielsweise verschiedene Sensordaten ausgelesen werden.

Client-Server-Architektur

Dieses Kapitel beschreibt die wesentlichen Konzepte und Ideen der Client-Server-Architektur sowie die verwendeten Programmiersprachen. Dabei wird auch auf deren Vor- und Nachteile eingegangen, sowie mögliche Alternativen diskutiert.

Frameworks und Programmiersprachen

Die erste Version der Client App wurde mit purem JavaScript, HTML und CSS geschrieben. Es kam kein Framework zum Einsatz. Dadurch konnten zwar recht schnell gute Ergebnisse erzielt werden, die Wartbarkeit und die Erweiterbarkeit waren jedoch nur bedingt gegeben. Für den Node-Server als Backend verhielt es sich ähnlich.

In einer zweiten Iteration wurden dahingehend größere Änderungen vorgenommen. Der Node-Server stellt keine klassische REST-API zur Verfügung, sondern setzt auf die noch relativ neue GraphQL-Technologie. Bei einer GraphQL-API erhält der Client mehr Kontrolle über seine API-Requests. Die API wurde mit Apollo-Framework umgesetzt. Apollo stellt zudem automatisch einen GraphQl-Playground zur verfügung, mit dem die Schnittstelle auf einfache Art und Weise und ohne externe Tools getestet werden kann.

Für die Client-App wird das JavaScript-Framework Vue.js genutzt. Alternativ wären auch andere JS-Frameworks, wie zum Beispiel React oder Angular denkbar gewesen. Vue.js zeichnet sich aber vor allem durch eine schnelle Erlernbarkeit aus und bietet so einen einfacheren Einstieg für noch wenig erfahrene Entwickler.

Die Verwendung eines Frameworks hat zudem das gleichzeitige Arbeiten an mehreren Interface-Prototypen deutlich vereinfacht.

Implementierung

Für den Einsatz im Bus wird ein iPad Mini genutzt, welches mit einer speziellen Halterung fest im Bus installiert ist und über den Bus durchgehend mit Strom versorgt wird. Der Anwendungsserver ist mit Node.js implementiert und stellt eine GraphQL-API nach außen zur Verfügung. Neben regulären HTTP-Requests ist auch eine Kommunikation über einen Websocket möglich. Dadurch können Nachrichten ohne die durch ein Polling entstehende Verzögerung direkt vom Server an den Client gesendet werden. Dies ist vor allem dann sinnvoll und auch notwendig, wenn die Client-App auf Echtzeitdaten angewiesen ist.

Durch die strikte Trennung von Client und Server kann die GraphQL-API natürlich auch ohne Probleme in Kombination mit einer anderen, selbst entwickelten, Client-App genutzt werden.

Der Node-Server erhält die Fahrdaten der Busse über eine Redis-Instanz. Für jeden Bus existiert ein separates *Topic*, in welchem die Fahrdaten gepublisht werden. Über die GraphQL-API kann jeder Client dann entsprechend das für ihn relevante *Topic* subscriben. Durch die Nutzung von Redis spielt die Herkunft der Daten keine Rolle, es muss nur sichergestellt werden, dass die Fahrdaten in einem bestimmten Format im entsprechenden Topic gepublisht werden. In diesem konkreten Anwendungsfall werden die Fahrdaten vom Bus an den Hersteller gesendet und dort in einer MySQL-Datenbank gespeichert. Diese Datenbank wird auf einen Server der Universität zu Lübeck gespiegelt. Zusätzlich läuft ein Node-Script, welches ankommende Daten aufbereitet und schlussendlich im Redis-Topic veröffentlicht und so dem NuR.E-Anwendungsserver zur Verfügung stellt. Für die Nutzung des Codes muss dementsprechend möglicherweise zunächst ein Skript geschrieben werden, welches die Fahrdaten über Redis dem NuR.E-Anwendungsserver zur Verfügung stellt. Redis dient somit als Brücke zwischen dem Anwendungsserver und den Datenquellen. Auch die Simulation ist im Wesentlichen nur eine

weitere Datenquelle und published die Fahrdaten im entsprechenden Topic.

Datenbankarchitektur

Auf dem MySQL-Server liegen drei Datenbanken: *vehicleData*, *processedVehicleData* und *nureData*. Die vehicleData-Datenbank bildet das Gegenstück zur Datenbank des Busherstellers. Alle Fahrdaten werden automatisch in diese Datenbank übertragen. Die Daten werden bereinigt, aufbereitet und anschließend in der processedVehicleData-Datenbank gespeichert. Diese beiden Datenbanken sind ausschließlich für die Datenübertragung notwendig und werden gegebenenfalls nicht benötigt, sofern die Daten auf eine andere Art und Weise übertragen werden. Die hier zum Einsatz kommende Art der Datenübertragung ist in keinster Weise optimal. Durch die IT-Infrastruktur des Busherstellers gab es jedoch keine umsetzbaren Alternativen.

Viel wichtiger für das NuR.E-System ist die nureData-Datenbank. In dieser werden systemrelevante Daten abgelegt. Dazu zählen unter anderem Benutzerdaten, von dem Fahrpersonal über das Tablet abgegebenes Feedback und vom Tablet aufgezeichneten Sensordaten.