Wetterstation

Projekthandbuch und -dokumentation

Ing. Hannes Aurednik

Eva Gergely

Salko Nuhanovic

Stephan Nöhrer

im Zuge der Lehrveranstaltung Embedded Systems Engineering

im Sommersemester 2018

betreut durch: Patrick Schmitt MSc

Einleitung

In folgendem Dokument wird das Projekt „Wetterstation“, welches im Zuge der Lehrveranstaltung Embedded Systems Engineering entwickelt wurde detailliert beschrieben und dokumentiert.

Statt einer persönlichen Lerndokumentation wurde in diesem Dokument versucht, eine Projektschilderung lege artis anzufertigen, die einerseits die Anforderungen der Lehrveranstaltung, andererseits aber auch den Interessen eventueller, zukünftiger Stakeholder genügt.

Zusätzlich wurde von den Verfassern Augenmerk auf die Verhältnismäßigkeit der Dokumentation zum eigentlichen Projekt gelegt.

# Inhaltsverzeichnis

[Inhaltsverzeichnis 3](#_Toc518321763)

[Projekthandbuch 5](#_Toc518321764)

[1. Projektauftrag 5](#_Toc518321765)

[2. Projektzieleplan 6](#_Toc518321766)

[3. Changerequests 7](#_Toc518321767)

[Change 1 7](#_Toc518321768)

[4. Projektstrukturplan 8](#_Toc518321769)

[5. Projektphasenplan 9](#_Toc518321770)

[6. Arbeitspaketspezifikationen 10](#_Toc518321771)

[7. Projektmeilensteinplan 15](#_Toc518321772)

[8. Projektfunktionendiagramm 16](#_Toc518321773)

[9. Projektregeln und Projektwerte 17](#_Toc518321774)

[10. Projektabschlussbericht 18](#_Toc518321775)

[Projektdokumentation 19](#_Toc518321776)

[1. Einleitung 19](#_Toc518321777)

[Projektziel 19](#_Toc518321778)

[2. AP: Projektkommunikation 19](#_Toc518321779)

[Verantwortlicher: Hannes Aurednik 19](#_Toc518321780)

[Besonderheiten Abschlusspräsentation 20](#_Toc518321781)

[Verwendete Tools 22](#_Toc518321782)

[3. AP: Frontend Design 22](#_Toc518321783)

[Verantwortlicher: Eva Gergely 22](#_Toc518321784)

[Verwendete Tools 23](#_Toc518321785)

[Screenshots der Mockups 24](#_Toc518321786)

[4. AP: Architektur 25](#_Toc518321787)

[Verantwortlicher: Salko Nuhanovic 25](#_Toc518321788)

[5. AP: Netzwerk 29](#_Toc518321789)

[Verantwortlicher: Hannes Aurednik 29](#_Toc518321790)

[Verwendete Tools 33](#_Toc518321791)

[6. AP: Sensorik 34](#_Toc518321792)

[Verantwortlicher: Eva Gergely 34](#_Toc518321793)

[Verwendete Tools 36](#_Toc518321794)

[Vorgehensweise 36](#_Toc518321795)

[7. AP: Backend 40](#_Toc518321796)

[Verantwortlicher: Stephan Nöhrer 40](#_Toc518321797)

[Verwendete Tools 41](#_Toc518321798)

[Verwendete Technologien 41](#_Toc518321799)

[Architektur und Patterns 42](#_Toc518321800)

[Vorgehensweise 42](#_Toc518321801)

[Lessons learned 46](#_Toc518321802)

[8. AP: Alexa Skill 47](#_Toc518321803)

[Verantwortlicher: Salko Nuhanovic 47](#_Toc518321804)

[Verwendete Tools: 56](#_Toc518321805)

[9. AP: Frontend Implementierung 56](#_Toc518321806)

[Verantwortlicher: Stephan Nöhrer 56](#_Toc518321807)

[Verwendete Tools 57](#_Toc518321808)

[JavaScript, CSS und Tools 58](#_Toc518321809)

[Vorgehensweise 58](#_Toc518321810)

[Lessons learned 61](#_Toc518321811)

[10. AP: Embedded Integration 61](#_Toc518321812)

[Verantwortlicher: Hannes Aurednik 61](#_Toc518321813)

[Analyse und Bewertung Integrationsszenarien 62](#_Toc518321814)

[Integration mittels Mailboxen 63](#_Toc518321815)

[Aufgetretene Herausforderungen 65](#_Toc518321816)

[Refactoring-Maßnahmen 65](#_Toc518321817)

[11. Installationsanleitung 67](#_Toc518321818)

[Anhang 68](#_Toc518321819)

[12. Verwendete Tools 68](#_Toc518321820)

[13. Abbildungsverzeichnis 69](#_Toc518321821)

[14. Quellenverzeichnis 70](#_Toc518321822)

# Projekthandbuch

|  |  |
| --- | --- |
| Projektauftrag | |
| **Starttermin:** | **Projektendtermin:** |
| 19.02.2018 | 02.07.2018 |
| **Projektziele** | **Nicht-Projektziele** |
| Entwicklung eines Smarthome-Assistent-Systems zur Temperatur und Luftfeuchtigkeitsinformation inkl. Userinteraktion über Amazon Echo. | Entwicklung von nicht explizit definierten Funktionen.  Unterstützung von WLAN Kommunikation der Mikrocontroller |
| **Projektauftraggeberteam** | **Projektmanager** |
| Patrick Schmitt MSc | Ing. Hannes Aurednik |
| **Projektteammitglieder** | |
| Ing. Hannes Aurednik  Eva Gergely  Salko Nuhanovic  Stephan Nöhrer | |

|  |  |
| --- | --- |
| Projektzieleplan | |
| **Hauptziele** | **Adaptiert per …** |
| Entwicklung eines Amazon Echo Alexa Skills zur Userinteraktion | 19.02.2018 |
| Implementieren eines Webservers zur Datenanzeige, -speicherung und Bereitstellung von Schnittstellen | 19.02.2018 |
| Implementieren einer per Netzwerk erreichbaren Messstation (kabelgebunden) | 19.02.2018 |
| Unterstützung mehrerer Messstationen | 19.02.2018 |
|  |  |
| **Zusatzziele (optional)** | **Adaptiert per …** |
| Entwickeln einer Dashboard Funktionalität mit Visualisierung des Werteverlaufes der Messdaten | 11.04.2018 |
|  |  |
|  |  |
| **Nicht-Ziele** | **Adaptiert per …** |
| Produktreife „Plug and Play – Lösung“ | 20.02.2018 |
| Implementieren einer per Netzwerk erreichbaren Messstation (WLAN) | 09.05.2018 |
|  |  |
|  |  |

Annahmen und Interpretation:

|  |  |
| --- | --- |
| Changerequests | |
| **Änderung** | **Adaptiert per …** |
| Keine relevanten Änderungen zum ursprünglichen Projektscope | 02.07.2018 |
|  |  |
|  |  |

|  |
| --- |
| Change 1 |
| **Ursprüngliches Ziel** |
|  |
| **Adaptiertes Ziel** |
|  |
| **Begründung** |
|  |

|  |
| --- |
| Projektstrukturplan |
|  |

Annahmen und Interpretation:

|  |
| --- |
| Projektphasenplan |
|  |

Annahmen und Interpretation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Arbeitspaketspezifikationen | | | | |
| **PSP-Code:** | 1.2.2 | **AP-Bezeichnung** | **Lösungsarchitektur** | |
| **AP-Inhalt** | | | | **Adaptiert per …** |
| * Entwicklung einer zweckmäßigen Lösungsarchitektur | | | | 19.02.2018 |
| **AP-Nicht-Inhalte** | | | | **Adaptiert per …** |
| * Entwicklung SW-Architektur für Embedded Module | | | | 19.02.2018 |
| **AP-Ergebnisse** | | | | **Adaptiert per …** |
| * Anforderungsgerechte Lösungsarchitektur | | | | 19.02.2018 |
| **AP-Leistungsfortschrittsmessung** | | | | **Adaptiert per …** |
| * Vollständigkeit Lösungsarchitektur * Konformität Lösungsarchitektur | | | | 19.02.2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PSP-Code:** | **1.1.1** | **AP-Bezeichnung** | **Projektkommunikation** | |
| **AP-Inhalt** | | | | **Adaptiert per …** |
| * Regelmäßige Kommunikation Projektergebnisse * Planung Abschlusspräsentation * Kommunikation Projektrisiken | | | | 19.02.2018 |
| **AP-Nicht-Inhalte** | | | | **Adaptiert per …** |
|  | | | |  |
| **AP-Ergebnisse** | | | | **Adaptiert per …** |
| * Erfolgreiche Abschlusspräsentation * Backup Video für Abschlusspräsentation | | | | 17.06.2018 |
| **AP-Leistungsfortschrittsmessung** | | | | **Adaptiert per …** |
| * Durchführung regelmäßiger Projektkommunikation zum Auftraggeber * Zeitgerechte Kommunikation abnahmeverhindernder Projektrisiken | | | | 19.02.2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PSP-Code:** | **1.3.4** | **AP-Bezeichnung** | **Embedded - Netzwerk** | |
| **AP-Inhalt** | | | | **Adaptiert per …** |
| * Implementieren einer architekturkonformen Netzwerkanbindung * Unterstützung TCP basierter Kommunikation * Unterstützung von kabelgebundenen Netzwerkverbindungen | | | | 19.02.2018 |
| **AP-Nicht-Inhalte** | | | | **Adaptiert per …** |
| * Unterstützung UDP basierter Kommunikation * Unterstützung von drahtlosen Netzwerkverbindungen | | | | * 19.02.2018 |
| **AP-Ergebnisse** | | | | **Adaptiert per …** |
| * Erfolgreiche Netzwerkkommunikation der Mikrocontroller | | | | 19.02.2018 |
| **AP-Leistungsfortschrittsmessung** | | | | **Adaptiert per …** |
| * Funktion der implementierten Netzwerkanbindung * Funktion der IPC Anbindung | | | | 19.02.2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PSP-Code:** | **1.3.3** | **AP-Bezeichnung** | **Sensorik** | |
| **AP-Inhalt** | | | | **Adaptiert per …** |
| * Implementieren der Sensorikfunktionalität * Implementieren einer Ausgabe auf der Konsole zwecks Fehlerüberprüfung | | | | 02.07.2018 |
| **AP-Nicht-Inhalte** | | | | **Adaptiert per …** |
| * Implementieren von Mailboxes | | | | 17.05.2018 |
| **AP-Ergebnisse** | | | | **Adaptiert per …** |
| * Sensorik funktionsfähig * Ausgaben auf der Konsole durch bedingtes Kompilieren abgrenzen | | | | 19.02.2018 |
| **AP-Leistungsfortschrittsmessung** | | | | **Adaptiert per …** |
| * Temperatur kann ausgelesen werden * Luftfeuchtigkeit kann ausgelesen werden | | | | 19.02.2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PSP-Code:** | **1.3.2** | **AP-Bezeichnung** | **Backend** | |
| **AP-Inhalt** | | | | **Adaptiert per …** |
| * Datenpersistenz im definierten Umfang * Implementieren der Serverschnittstelle zum Empfang der Sensordaten * Bereitstellung der für das Frontend notwendigen Funktionalitäten | | | | 19.02.2018 |
| **AP-Nicht-Inhalte** | | | | **Adaptiert per …** |
| * Security via API Key oder Benutzerregistrierung | | | | 19.02.2018 |
| **AP-Ergebnisse** | | | | **Adaptiert per …** |
| * Funktionierende RESTful APIs für Sensoren * Funktionierende RESTful APIs für Alexa * Deployment der Web App in Azure | | | | 17.05.2018 |
| **AP-Leistungsfortschrittsmessung** | | | | **Adaptiert per …** |
| * Funktionierende Datenpersistenz * Funktion der Serverschnittstelle * Funktion der vom Frontend benötigten Funktionalitäten | | | | 19.02.2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PSP-Code:** | **1.3.1** | **AP-Bezeichnung** | **Frontend Implementierung** | |
| **AP-Inhalt** | | | | **Adaptiert per …** |
| * Entwickeln einer Weboberfläche für Benutzerinteraktionen | | | | 19.02.2018 |
| **AP-Nicht-Inhalte** | | | | **Adaptiert per …** |
| * Voller Funktionsumfang auf Mobile Devices | | | | 19.02.2018 |
| **AP-Ergebnisse** | | | | **Adaptiert per …** |
| * Devices können erfasst, bearbeitet oder gelöscht werden * Echtzeitdaten der Devices werden angezeigt * Aktuellste Messdaten werden angezeigt | | | | 17.05.2018 |
| **AP-Leistungsfortschrittsmessung** | | | | **Adaptiert per …** |
| * Generelles Layout ist implementiert * Daten aus der Datenbank werden angezeigt * Benutzereingaben werden persistiert | | | | 19.02.2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PSP-Code:** | **1.3.5** | **AP-Bezeichnung** | **Alexa Skill** | |
| **AP-Inhalt** | | | | **Adaptiert per …** |
| * Entwickeln eines Alexa Skills welcher Messungen mittels REST Call abruft (nach Fire and Forget Prinzip) | | | | 19.02.2018 |
| **AP-Nicht-Inhalte** | | | | **Adaptiert per …** |
| * Konversation mit Alexa | | | | 19.02.2018 |
| **AP-Ergebnisse** | | | | **Adaptiert per …** |
| * Alexa Skill ist funktionsfähig * Dazugehörige Lambda Funktion ist deplyoed | | | | 22.05.2018 |
| **AP-Leistungsfortschrittsmessung** | | | | **Adaptiert per …** |
| * Messungen des TiviaC Boards sind via Alexa Skill abrufbar | | | | 02.06.2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PSP-Code:** | **1.2.1** | **AP-Bezeichnung** | **Frontend Design** | |
| **AP-Inhalt** | | | | **Adaptiert per …** |
| * Erstellung von Mockups, um das Hauptaugenmerk bei der Entwicklung des Frontend einzugrenzen * Auswahl eines geeigneten Tools * Registrierung, um Produkt verwenden zu können * Erstellung von Mockups im Tool BalsamiqCloud | | | | 18.03.2018 |
| **AP-Nicht-Inhalte** | | | | **Adaptiert per …** |
| * Details der Implementierung mit einzubeziehen | | | | 18.03.2018 |
| **AP-Ergebnisse** | | | | **Adaptiert per …** |
| * Mockups für die Hauptseite, für die Anzeige der verfügbaren Geräte und für die Anzeige von Echtzeitdaten erstellt | | | | 18.03.2018 |
| **AP-Leistungsfortschrittsmessung** | | | | **Adaptiert per …** |
| * Fertige Mockups, die den Vorstellungen aller Teammitglieder entsprechen | | | | 18.03.2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PSP-Code:** | **1.4.1** | **AP-Bezeichnung** | **Embedded Integration** | |
| **AP-Inhalt** | | | | **Adaptiert per …** |
| * Kommunikation der beiden embedded SW-Module sicherstellen | | | | 18.03.2018 |
| **AP-Nicht-Inhalte** | | | | **Adaptiert per …** |
|  | | | |  |
| **AP-Ergebnisse** | | | | **Adaptiert per …** |
| * Kommunikation zwischen den embedded Softwaremodulen ist möglich * Kommunikation zwischen den embedded Softwaremodulen ist synchronisieirt | | | | 18.03.2018 |
| **AP-Leistungsfortschrittsmessung** | | | | **Adaptiert per …** |
| * Komminikation zwischen den Modulen möglich | | | | 18.03.2018 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Projektmeilensteinplan | | | | |
| **Meilensteinbezeichnung** | **Plantermin** | **Plantermin adaptiert per …** | **Isttermin** |
| Projektplanung abgeschlossen | 02.04.2018 |  | 02.04.2018 |
| NW Kommunikation möglich | 15.06.2018 | 30.06.2018 | 30.06.2018 |
| HTU Sensorik fertiggestellt | 22.06.2018 |  | 22.06.2018 |
| Backend fertiggestellt | 10.06.2018 | 20.06.2018 | 20.06.2018 |
| Frontend fertiggestellt | 30.06.2018 |  | 30.06.2018 |
| Alexa Skill fertiggestellt | 22.06.2018 |  | 22.06.2018 |
| Embedded Integration fertiggestellt | 29.06.2018 | 30.06.2018 | 30.06.2018 |
| Prototyp präsentierbar | 30.06.2018 |  | 30.06.2018 |
| Dokumentation abgeschlossen | 01.07.2018 | 02.07.2018 | 02.07.2018 |
| Projekt abgenommen | 02.07.2018 |  | 02.07.2018 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Projektfunktionendiagramm | | | | | | | |
| **PSP-Code** | **AP-Bezeichnung** | **Rollen und Umwelten** | | | | | |
| **Projektauftrag-geberteam** | **Aurednik** | **Gergely** | **Nuhanovic** | **Nöhrer** | **Sonstige** |
|  | Projektkommunikation | **I** | **D** | **Z** | **B** | **Z** |  |
|  | Frontend Design | **I** | **Z** | **D** | **Z** | **B** |  |
|  | Architektur | **I** | **Z** | **Z** | **D** | **B** |  |
|  | Netzwerk | **I** | **D** | **Z** | **B** | **B** |  |
|  | Sensorik | **I** | **B** | **D** | **Z** | **Z** |  |
|  | Backend | **I** | **Z** | **ZI** | **B** | **D** |  |
|  | Dokumentation | **I** | **D** | **D** | **D** | **D** |  |
|  | Alexa | **I** | **Z** | **Z** | **D** | **Z** |  |
|  | Frontend Implementierung | **I** | **Z** | **B** | **Z** | **D** |  |
|  | Embedded Integration | I | D | Z | I | B |  |

Nach IBZED-Schema:

* I - Information:
* B - Beratung:
* Z - Zustimmung
* E - Entscheidung
* D - Durchführung:

Annahmen und Interpretation:

|  |  |
| --- | --- |
| Projektregeln und Projektwerte | |
| **Projektregeln** | **Adaptiert per …** |
| Regelmäßige JF werden pünktlich begonnen und beendet | 19.02.2018 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **Projektwerte** | **Adaptiert per …** |
| Offenheit und Transparenz werden gelebt | 19.02.2018 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Annahmen und Interpretation:

|  |
| --- |
| Projektabschlussbericht |
| **Gesamteindruck: Projekt** |
|  |
| **Reflexion: Projektziele** |
|  |
| **Reflexion: Projektleistungsfortschritt** |
|  |
| **Reflexion: Projekttermine** |
|  |
| **Reflexion: Projektkosten, Projektressourcen** |
|  |
| **Reflexion: Projektumwelten, Beziehungen zu anderen Projekten** |
|  |
| **Reflexion: Projektorganisation und Projektkultur** |
|  |
| **Zusammenfassende Erfahrungen** |
|  |

# Projektdokumentation

|  |  |
| --- | --- |
| Einleitung | |
|  | Im Zuge des Projektes Wetterstation wurde von den beteiligten StudentInnen eine Lösung erarbeitet die in diesem Abschnitt genauer beschrieben wird.  Der Abschnitt Projektdokumentation dient insbesondere dazu, die technische Lösung zu beschreiben, eine Installationsanleitung bereitzustellen und die persönlichen Lernergebnisse der Projektmitglieder zu dokumentieren. Projektziel Das Ziel des Projektes war die Entwicklung einer Smart-Home Lösung zum Raumklima-Monitoring mittels Alexa in einem typischen Wohnumfeld. |

|  |  |
| --- | --- |
| AP: Projektkommunikation | |
|  | Verantwortlicher: Hannes Aurednik Das Arbeitspaket Projektkommunikation umfasste alle Tätigkeiten die eine effiziente Kommunikation innerhalb des Vorhabens sowie durch das Projekt nach außen gewährleisten.  Insbesondere galt es hier die folgenden Punkte sicherzustellen:   * Regelmäßige Statusupdates an den Projektauftraggeber * Koordination und Planung der internen Jours fixes * Sicherstellen des Informationsflusses im Projektteam * Planung und Vorbereitung der Abschlusspräsentation   Während für die ersten drei Punkte obiger Liste, aufgrund der geringen Anzahl an Projektmitgliedern, keine besonderen und erwähnenswerten Maßnahmen, wie beispielsweise ein expliziter Kommunikationsplan, zu setzen waren, erforderte die Abschlusspräsentation besonderes Augenmerk.  Dies ist primär durch die besondere netzwerktechnische Situation in den Räumlichkeiten des FH-Technikums induziert.  Anders als in einem Heimnetzwerk, für welches das Projekt konzipiert wurde, ist der Netzwerkzugang in der Fachhochschule strenger geregelt und die Mikrocontroller erhalten keine IP-Adressen.  Auch das direkte Verbinden eines handelsüblichen WLAN-Routers mit dem Netzwerk der FH, um auf diesem Wege eine Internetverbindung für die verwendeten Geräte zu erhalten, ist aus selbigem Grund nicht möglich.  Da eine funktionierende Netzwerkverbindung für die Sensorik-Komponenten jedoch von essentieller Bedeutung ist musste eine Alternativlösung gefunden werden. |

|  |  |
| --- | --- |
| Besonderheiten Abschlusspräsentation | |
|  | Um den Besonderheiten der Präsentationsumgebung Rechnung zu tragen wurden für die abschließende Vorführung mehrere Strategien entwickelt. Mangel an Temperaturunterschieden Um trotz der, aus dem Blickwinkel des Raumklimas, homogenen Umgebung der Fachhochschule unterschiedliche Temperatur- und Luftfeuchtigkeitswerte erzeugen und darstellen zu können ohne die zur Verfügung gestellte Hardware zu gefährden, wird im Zuge der Abschlusspräsentation ein Mikroklima geschaffen.  Dieses wird mittels handelsüblicher Kühlkörper in einer Kühltasche für den Medikamententransport erzeugt. Das verwendete Behältnis wurde gewählt da es ein weit geringes Volumen als vergleichbare Produkte aufweist und somit in einer kürzeren Zeitspanne die benötigten Umgebungsparameter erzeugt werden können. Netzwerk Eine größere Herausforderung stellten die eingeschränkten Möglichkeiten der Netzwerkumgebung dar.  Im Zuge mehrerer Projektsitzungen konnte aber, durch eine Erhebung der im Kreis der Projektmitglieder vorhandenen Hardwareressourcen eine Lösung entwickelt werden, die keine zusätzlichen Kosten verursacht und sämtliche Anforderungen für eine Live-Demonstration erfüllt, ohne ein Sicherheitsrisiko für das Netzwerk der Fachhochschule zu erzeugen.  Durch die Kombination eines UTMS- Routers und eines gebräuchlichen WLAN-Access Points mit Switch-Funktionalität auf mehreren RJ45 Ports konnte die folgende Netzwerktopologie erzeugt werden. |



Abbildung 1: Netzwerktopologie Abschlusspräsentation

|  |  |
| --- | --- |
|  | Obige Abbildung zeigt die Netzwerktopologie für die Abschlusspräsentation. Durch den UMTS Router wird dem dahinterliegenden Access Point eine Internetverbindung zur Verfügung gestellt die dieser seinerseits per LAN Schnittstellen an die beiden Sensoren und per WLAN an das Amazon Echo Device weiterreicht.  Der Server ist über das Internet erreichbar auf Microsoft Azure deployed.  Benutzerdevices wie Notebooks oder Tablets können sich über eine aktive Internetverbindung mit dem Server verbinden und die aktuellen Daten anzeigen sowie Sensoren konfigurieren.  Das Amazon Echo Device greift über die per WLAN und UMTS Router zur Verfügung gestellte Internetverbindung auf den Server zu. |

|  |  |
| --- | --- |
| Verwendete Tools | |
| Für das Arbeitspaket wurden folgende Tools verwendet:   * + Microsoft Powerpoint (g) |

|  |  |
| --- | --- |
| AP: Frontend Design | |
|  | Verantwortlicher: Eva Gergely Um sich im Vorfeld einen klaren Überblick über die Requirements an das Frontend zu verschaffen, wurden Mockups erstellt, die jeweils die drei wichtigsten Oberflächen darstellen.  Zunächst kommt man auf die Hauptseite, wo man mittels Klick die gewünschte Funktionalität auswählen kann. Es steht ein Menüpunkt für die Anzeige eines generellen Überblicks über die bis dato gesammelten Daten zur Verfügung, ein Menüpunkt für die Anzeige der Echtzeitdaten, sowie ein Menüpunkt um die erfassten Geräte anzeigen zu können.  Auf Mockup Nr. 2 wird die Anzeige der Echtzeitdaten dargestellt, die durch ein einfaches Diagramm verwirklicht werden soll.  Auf Mockup Nr. 3 sieht man die Oberfläche für die Anzeige der erfassten Geräte. Es soll die Nummer des Geräts, die MAC-Adresse, das Erfassungsdatum sichtbar sein und ob das Gerät aktiv ist. Es soll außerdem möglich sein, die Liste zu editieren. |

|  |  |
| --- | --- |
| Verwendete Tools | |
| Für das Frontend Design wurde das Tool Balsamiq verwendet.   * + Balsamiq (c)   <https://balsamiq.cloud/>  Eine Webanwendung, mit der man sog. Mockups erstellen kann. |

### Screenshots der Mockups

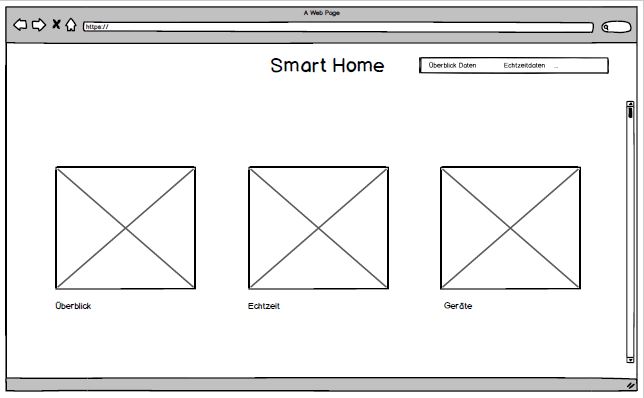


Abbildung 2: Mockup Hauptseite

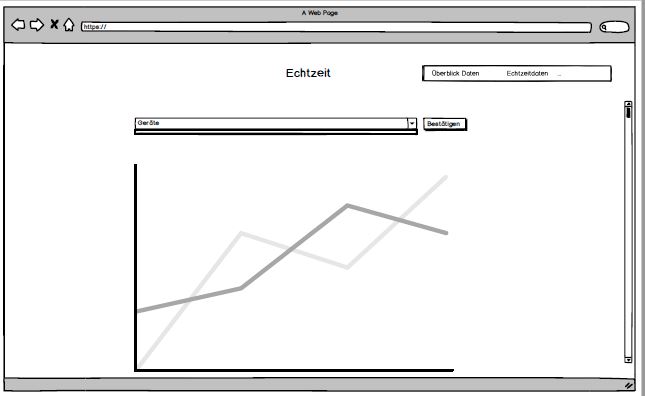


Abbildung 3: Mockup Echtzeitdaten

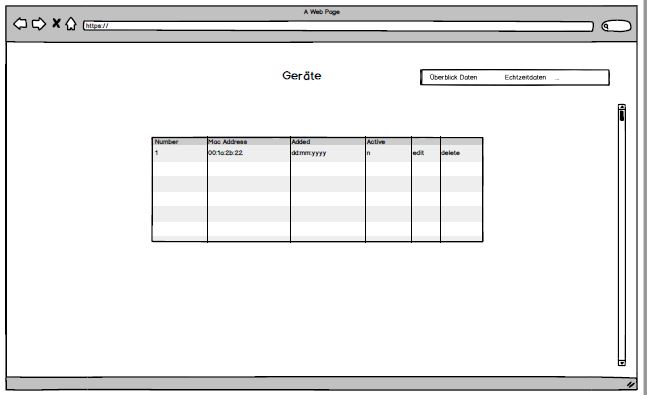
**

Abbildung 4: Mockup Geräteauswahl

|  |  |
| --- | --- |
| AP: Architektur | |
|  | Verantwortlicher: Salko Nuhanovic Der folgende Abschnitt befasst sich mit der Gesamtlösungsarchitektur der einzelnen Module und Komponenten. |

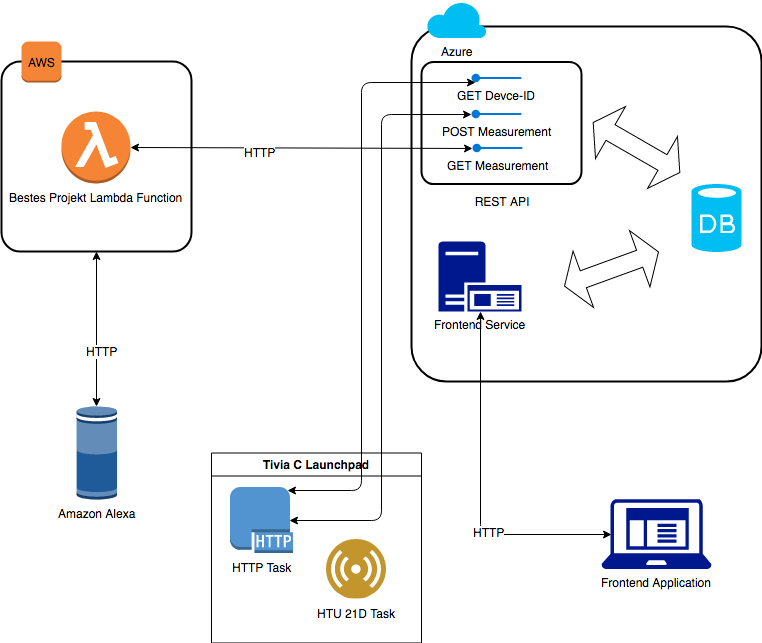


Abbildung 5: Architektur

|  |  |
| --- | --- |
|  | Die Architektur unseres Projekts setzt sich aus den folgenden Komponenten zusammen:   * Alexa Skill * Tivia C Launchpad (inkl. HTU 21D Click Modul) * Backend Application (inkl. Web Dashboard)  Backend Mittelpunkt des Projekts ist die Backendapplikation, welche Alexa Skill, Applikation Server (Web Applikation) und µController verbindet und auf Azure deployed ist.  Bestandteile der Backendapplikation sind:   * Datenbank:   + Enthält aktuelle Messungen und dazugehörigen registrierten Geräte * Applikation Server   + REST API - stellt folgende Endpoints zur Verfügung:     - Vergabe von IDs für Wetterstationen     - Annahme von Messwerten     - Liste der registrierten Wetterstationen     - Auskunft über aktuelle Messwerte   + Dashboard – ermöglicht es dem User Messwerte einzusehen und Wetterstationen zu registrieren   Die REST API ist ein Webservice und ist somit via HTTP aufrufbar. Alexa Skill Der Alexa Skill besteht aus einem Interaction Model und aus einer Lambda Funktion welche auf AWS deployed sind.  Das Interaction Model ist das Interface für den User welches auf bestimmte Schlagwörter hört und entsprechend Logik durchführt.  Lambda ist ein eine Plattform welche FaaS (Function as a Service) anbietet, somit kann einfach ein Code Snippet nach Lambda deployed werden und dann remote ausgeführt werden. µController Der Micro Controller hat einen HTU21D Temperatur und Luftfeuchtigkeit Sensor welche periodisch eine Messung durchführt. Zusammenspiel der einzelnen Komponenten Grundlegend kann man sagen das alle Komponenten der Architektur via HTTP miteinander kommunizieren.  Damit eine Wetterstation (Tivia C Board) Messungen an die Backendapplikation senden kann, muss dieses zuerst registriert werden, dies passiert über das Web Dashboard.  Daraufhin kann der µController Messungen seiner Umwelt durchführen und diese an die Backendapplikation via HTTP POST übergeben. In Azure angelangt werden die Messungen in der Datenbank gespeichert. Die Messungen sind dann über zwei Kanäle aufrufbar:   * Dashboard * Alexa Skill   Der Alexa Skill bekommt seine Daten indem er die Lambda Funktion ausführt, diese ruft dann die Backendapplikation via HTTP GET auf und bekommt die Messungen. Anschließend verarbeitet Alexa das Ergebnis und liest es vor. Further Work Da zum Schluss noch ein bisschen Zeit übriggeblieben ist, wurde überlegt wie die Architektur verbessert werden kann. Nach Recherche hat sich ergeben das man einen MQTT Broker im Backend aufnehmen könnte um die Messungen entgegen zu nehmen, dieser würde dann die HTTP Kommunikation zwischen Wetterstation und Backend ersetzen.  Dies könnte durchgeführt werden indem man einen MQTT Client in der Wetterstation einbaut, welche dann einfach Messungen in den Broker published. Der Consumer auf der anderen Seite kann dann diese Daten auswerten. Lessons Learned: MQTT hat folgende Vorteile gegenüber HTTP:   * Schneller * Mehr Durchsatz * Weniger Energieverbrauch * Weniger Bandbreite |

|  |  |
| --- | --- |
| AP: Netzwerk | |
|  | Verantwortlicher: Hannes Aurednik Das Arbeitspaket Netzwerk umfasst die Analyse der typischen Charakteristika in Heimnetzwerken sowie die Entwicklung der Netzwerk-Kommunikationsfunktionalität der Mikrocontroller. Analyse gebräuchlicher Netzwerkumgebungen Die Analyse typischer Heimnetzwerke ergab, dass im heutigen Regelfall zumindest ein WLAN- Router im Netz existiert. Bei diesem handelt es sich zumeist um ein vom Internet Service Provider zur Verfügung gestelltes Produkt. Vereinzelt sind auch Access Points von verschiedenen Herstellern für den Endkunden-Massenmarkt zu finden.  Beide Geräteklassen verfügen in der Regel über mehrere LAN-Ports mit Switching-Funktionalität und sind in der Lage als DHCP Server zu fungieren.  Die erhobenen Informationen zeigen somit keinen gesonderten Entwicklungsbedarf oder Aufwand um die angestrebte Lösung umsetzen zu können.  Abbildung 6 beschreibt den Netzwerkaufbau der im Zuge des Projektes entwickelten Lösung.  Die beiden Sensoren sind per Kabel mit dem Access Point verbunden. Das Amazon Echo Device kommuniziert per WLAN.  Der Server, welcher die Werte empfängt und per Schnittstelle zum Alexa Skill zu Verfügung stellt, kann sowohl im Heimnetz als auch außerhalb deployed werden.  Dies kann durch Änderungen im Alexa Skill konfiguriert werden.  Für die volle Funktionalität ist in jedem Fall eine aktive Internetverbindung notwendig, da das Alexa Device sonst nicht mit den dahinterliegenden Cloudservices kommunizieren kann. Embedded Netzwerk Lösung Die implementierte Lösung basiert auf TCP/IP und kommuniziert mittels einem vordefinierten JSON Format mit dem Server.  Als Basis für die Software dient das httpget Beispiel von TI-RTOS, welches in der Umsetzung adaptiert und um weitere Module ergänzt wurde.  Der HTTP GET Request wird verwendet um nach dem initialen Programmstart das Device mittels MAC Adresse am Server anzumelden.  Dazu wird nach erfolgreichem Erhalt einer IP-Adresse die MAC Adresse des Devices ausgelesen und mittels einer eigenen Funktion an die zu erfragende URL angefügt. [1]  Der Server beantwortet diesen GET mit einer Device-ID welche der Controller speichert und zur Identifikation in allen weiteren Nachrichten an den Server anführt.  Anschließend wechselt der Task in einen „Listening“-Modus in dem er auf Werte von den Sensorik Modulen wartet.  Wenn der http-Task über die Mailboxen die gewünschten Werte erhält werden diese in einen JSON String gepackt und als Payload mit einem http POST an den Server gesendet. [2]  Im Fall eines erfolglosen Requests bricht die Applikation ab, da davon ausgegangen werden muss, dass entweder der Server oder das Netzwerk von einer Störung betroffen sind.  Auch wenn das Device nicht vorab am Server registriert wurde kommt es zu einem Fehler der das Programm abbricht. Dies wurde umgesetzt um sicherzustellen, dass nur berechtigte Devices Werte an einen Server senden können.  Nach erfolgreichem POST wartet der Controller wieder auf aktuelle Temperatur- und Luftfeuchtigkeitswerte.  Dieser Ablauf wird auch in Abbildung 7 schematisch dargestellt. Testing Um im Zuge der Entwicklung die Abhängigkeiten zu den anderen Modulen, speziell zur Serverimplementierung, gering zu halten wurden frei verfügbare Mock-Tools verwendet die gesendete GET- und POST Requests anzeigen.  Diese sind im Punkt Verwendete Tools dokumentiert. |

|  |  |
| --- | --- |
| Future Work | |
|  | In zukünftigen Projekten kann versucht werden die Mikrocontroller mit Hilfe eines WLAN-Moduls auch drahtlos zu verbinden und so mehr Flexibilität für den Benutzer zu bieten.  Im Projekt wurde dieser Schritt diskutiert aber aufgrund von Ressourcenmangel nicht weiterverfolgt  Ein weiterer Punkt für zukünftige Arbeit wäre das Erstellen des JSON Strings mittels der, für embedded Devices entwickelten Library JSMN. Dabei handelt sich um einen leichtgewichtigen JSON Parser der einfach in Embedded Projekte integriert werden kann.  Dieser wurde bereits in das Projekt importiert konnte aus Zeitmangel aber nicht mehr voll integriert werden. |

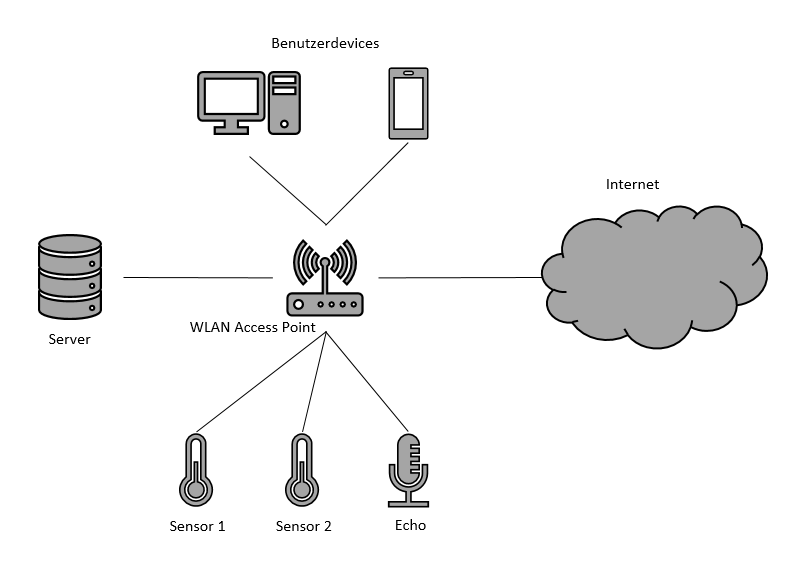


Abbildung 6: Netzwerktopologie Betriebszenario

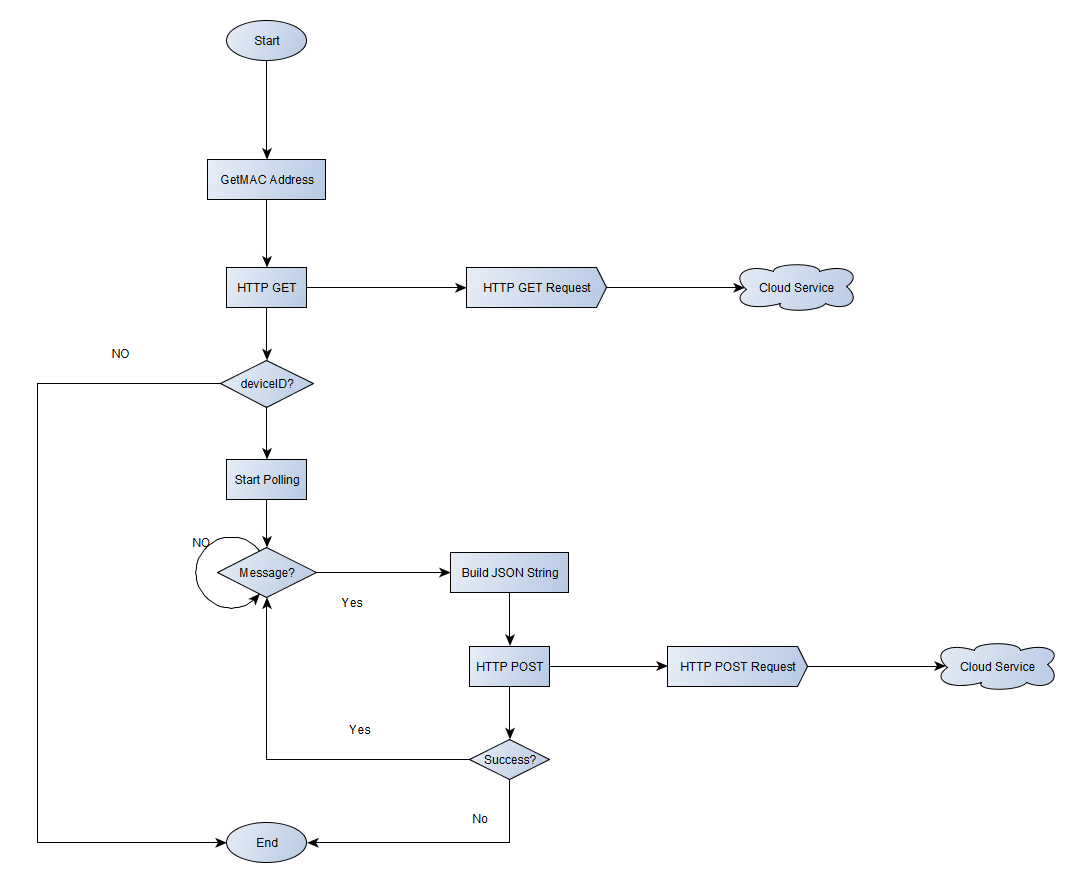


Abbildung 7: Ablaufdiagramm http Task

|  |  |
| --- | --- |
| Verwendete Tools | |
|  | Für das Arbeitspaket wurden folgende Tools verwendet:  https://webhook.site (a)  Frei verfügbare Seite zum Testen von Webhooks und HTTP Requests  Erstellt eine eindeutige URL gegen die man seine Implementierung teste kann und welche einem die empfangenen Request inkl. Daten anzeigt.  Im Projekt verwendet um die HTTP POST und GET Funktionalität zu testen ohne, dass das eigentliche Service laufen muss.  Zusätzlich erlaubt die Verwendung des Services eine genauere Inspektion der von der selbst implementierten Methoden versendeten Requests.  <https://apitester.com/> (b)  Frei verfügbares Tool zum Erstellen und Versenden verschiedener HTTP Requests. Erlaubt somit das Testen bestehender Endpunkte.  Im Projekt verwendet um die Erreichbarkeit und Funktion des eigenen Servers zu testen.  Code Composer Studio (c)  Version: 7.2.0.00013  IDE von Texas Instruments für die Entwicklung von Embedded Projekten  Verwendet zum Entwickeln und Debuggen des Codes für den Mikrocontroller |

|  |  |  |
| --- | --- | --- |
| AP: Sensorik | | |
|  | Verantwortlicher: Eva Gergely Bei dem HTU21D Sensor handelt es sich um einen digitalen relativen Feuchtigkeitssensor mit Temperaturausgabe, der dem Plug-and-Play Ansatz folgt. Das Modul bietet kalibrierte, linearisierte Signale, die im digitalen I2C Format ausgelesen werden können. Jeder Sensor wird individuell kalibriert und getestet.  Die Verbindung mit dem Mikrocontroller erfolgt auf einfache Art und Weise durch die modularen, digitalen Feuchtigkeits- und Temperaturausgänge. Der Messbereich umfasst -40 Grad Celsius bis +125 Grad Celsius und 0 bis 100 % relative Feuchtigkeit (relative humidity).  Mögliche Anwendungsbereiche sind unter anderem in der Automobilindustrie, im medizinischen Bereich, bei Luftbefeuchter und in der Wetterbeobachtung.  Der I2C Bus operiert auf zwei bidirektionalen Leitungen, von denen eine mit SDA (serial data) und die andere mit SCL (serial clock) bezeichnet wird. Über SDA findet die tatsächliche Datenübertragung statt. Über SCL werden die Takt-Impulse gesendet. Die I2C-Bausteine lassen sich über ein 7- (oder 10-) Bit breites Adress-Byte selektieren.  Der I2C Bus basiert auf einer Master-Slave-Kommunikation. Der Master sendet eine Start-Condition, wodurch die Slaves angesprochen werden. Sie vergleichen ihre Adresse mit der vom Master spezifizierten Adresse und der angesprochene Slave und der Master können nun eine Kommunikation beginnen.    Die Kommunikation ist folgendermaßen aufgebaut: es wird die Start- oder Repeated Start-Condition gesendet. Darauf folgt die Adresse des angesprochenen Slaves und ein Read/Write Flag. Der Slave bestätigt nun an den Master, dass er bereit ist und der Master kann mit dem Auslesen der Daten fortfahren. Der Mikrocontroller „spricht” mit dem Slave. Möchte der Master weitere Daten lesen, sendet er ein ACK an den Slave. Sonst sendet er kein ACK, sondern abschließend eine Stop-Condition.    Der HTU21D Sensor benötigt eine Betriebsspannung zwischen 1,5V und 3,6V. Nach Inbetriebnahme braucht er höchstens 15ms, um den idle state zu erreichen. Während dieser Zeit muss SCK hoch bleiben. Vor dieser Zeit soll kein Befehl gesendet werden. Um eine einwandfreie Übertragung zu erleichtern, ist es empfohlen, nach Erreichen des idle Zustandes einen Soft Reset durchzuführen. | |
| Verwendete Tools | | |
| Für das Arbeitspaket wurden folgende Tools verwendet:   * + Code Composer Studio (c)   Version: 7.2.0.00013  IDE von Texas Instruments für die Entwicklung von Embedded Projekten  Verwendet zum Entwickeln und Debuggen des Codes für den Mikrocontroller | |
| Vorgehensweise | |
| Nach Erhalt des Mikrocontrollers und Sensors begann ich zunächst mit einer ausführlichen Recherche zur Funktionalität des Sensors. Danach musste ich meine Kenntnisse über I2C auffrischen, die bis dahin nur theoretischer Natur waren.    Problem: Davor noch nie mit I2C gearbeitet  Lösung: Beispielimplementierungen aus dem Internet und aus dem letzten Semester heranziehen und versuchen, die Funktionalität nachzuvollziehen.  Nachdem es in der Gruppe diskutiert wurde, wie die Weboberfläche aussehen sollte, erstellte ich Mockups. Dazu musste ich mich auf der Webseite Balsamiq registrieren und eine 30-tägige „Probemitgliedschaft“ abschließen. Weiters musste ich mich mit der Benutzung der Webseite auseinandersetzen.  Anschließend schaute ich mir Videos im Internet zum HTU21D Sensor an und unternahm erste Versuche, die I2C Funktionen, die für das Ein- und Auslesen der Daten benötigt werden, zu programmieren.  Der Beispielcode zum HTU21D Sensor, den ich von der Seite des Herstellers bezogen hatte, half beim Verständnis und bei der Rekonstruktion des grundsätzlichen Ablaufes der Verarbeitung der Daten.  Die Aufgabe bestand in weiterer Folge darin, die im Beispielcode enthaltenen Funktionen so umzusetzen, dass diese mit dem von uns verwendeten Mikrocontroller funktionieren.  Anhand der Codebeispiele auf der Herstellerseite von Texas Instruments, sowie anhand der Beispielimplementierungen aus dem Wintersemester 2018 gelang es mir relativ rasch, die Grundgerüste der ersten Funktionen I2CInit und I2CTransferConfig zu programmieren und die Initialisierung des I2C-Busses und eine Fehlerbehandlung einzubauen.  Der nächste Schritt war das Erstellen eines Grundgerüstes der Funktion setup\_Poll\_Task() - jedoch noch ohne tatsächliche Funktionalität.  Als nächstes wurde die Funktion HTU21DSoftReset entwickelt, welche für ein power off und anschließendes power on des Devices verwendet wird. Es wird empfohlen, die Funktion beim Start des Sensors durchzuführen, um diesen zu rebooten. Es wird 0xFE ins Register geschrieben (11111110), woraufhin das HTU21D Sensor-System neu initialisiert und die Operation gestartet wird.  Problem: Funktion im Beispielcode des Sensors vorhanden, jedoch nicht für den verwendeten Mikrocontroller geeignet.  Lösung: Durch Setzen von Breakpoints und Recherche auf verschiedenen Foren zu Texas Instruments konnte letztendlich ein zufriedenstellendes Grundgerüst gefunden werden.  Im nächsten Schritt wurden die Funktionen HTU21D\_REG\_read und HTU21D\_REG\_write erstellt.  Anschließend wurde die Funktion CalcValues fertig gestellt. In der Funktion werden die Werte aus dem Buffer übergeben und diese ausgelesen. Der so ermittelte Wert wird dann in der jeweiligen Funktion GetTemperature oder GetRelativeHumidity mittels der Berechnung aus der HTU21D Beispielimplementierung in den tatsächlichen Temperatur- / Luftfeuchtigkeitswert umgewandelt.  Für die Funktionen GetTemperature und GetRelativeHumidity wurde einerseits der Beispielcode des HTU21D Sensors herangezogen, andererseits Recherche im Internet durchgeführt. Um die Fehlersuche zu erleichtern, wurden Ausgaben auf der Konsole implementiert. Beim Debugging konnte die korrekte Erfassung der Werte bestätigt werden.  Problem: Beim Debugging war klar ersichtlich, dass die richtigen Werte erfasst werden konnten, jedoch wurde als Ergebnis die Null ausgegeben. Screenshots zum beschriebenen Problem werden weiter unten angehängt.  Lösung: Nach mehreren Debugging Versuchen, Recherche im Internet und Versuchen, Funktionen für die Konvertierung der float Werte zu schreiben, konnte festgestellt werden, dass es sich beim Problem um eine nicht inkludierte Library handelte.  Nachdem diese Library eingebunden wurde, lieferte die Ausgabe der Werte ein zufriedenstellendes Ergebnis.  In weiterer Folge arbeitete ich an den einzelnen Funktionen weiter und recherchierte die logischen Grundlagen hinter der jeweiligen Funktionalität.  Als letztes passte ich die Headerfiles an und versah die jeweiligen Funktionen mit Briefings. Ich führte auch einen Code Refactoring aus und setzte mich mit bedingter Kompilierung auseinander, um die Ausgaben auf der Konsole für Debugging-Zwecke behalten zu können. Future Work Auf die Implementierung eines CRC-Checks wurde aus Zeitgründen verzichtet. Als Future Work wäre es wünschenswert, einen CRC-Check zur Überprüfung der Korrektheit der Werte zu implementieren.  Screenshots:    Abbildung 8: HTU21D Nullwerte bei Ausgabe mit Debugging Hilfen | |

Abbildung 9: HTU21D Variablenwerte



Abbildung 10: Debugging Hilfe



Abbildung 11: Korrekte Variablenwerte

|  |  |
| --- | --- |
| AP: Backend | |
|  | Verantwortlicher: Stephan Nöhrer Das Backend stellt den Data Access Layer mit Persistenz Framework und Datenmodell für die RESTful APIs und die im Frontend angesiedelten Benutzerinteraktionen zur Verfügung. Die APIs dienen einerseits den Sensoren zur Übermittlung von Messdaten, andererseits der Alexa Solution zur Abfrage der aktuellsten Daten.  Logisch getrennt werden in dem Projekt Frontend und Backend im Sinne einer Client/Server Architektur, wobei diese Form der Trennung naturgemäß schwierig ist und es daher mitunter zu Überschneidungen in der Dokumentation kommen kann. Die aufgrund des gewählten MVC-Patterns implementierten Models und Controller werden jedenfalls dem Backend zugeordnet. Das Backend bildet somit das Herzstück der Web App und enthält die Business-Logik entsprechend den Anforderungen. |

|  |  |
| --- | --- |
| Verwendete Tools | |
| Für das Arbeitspaket wurden folgende Tools verwendet:   * + Visual Studio 2017 (d)   Version: 15.7.4  IDE von Microsoft   * + SQL Operations Studio (e)   Version: 0.24.1  Zum Verwalten von Azure SQL Databases |
| Verwendete Technologien | |
| Für das Arbeitspaket wurden folgende Technologien verwendet:   * + ASP.NET Core 2.0 (with .NET Core)   + ASP.NET MVC Core   + .NET Core Native DI   + Entity Framework Core 2.1   + Azure Cloud   + [Swashbuckle Swagger](https://github.com/domaindrivendev/Swashbuckle)   + [AutoMapper](https://automapper.org/)   + [MediatR](https://github.com/jbogard/MediatR)   + [Serilog](https://serilog.net/)   + [OdeToCode](https://github.com/OdeToCode/AddFeatureFolders) |

|  |  |
| --- | --- |
| Architektur und Patterns | |
| Für das Arbeitspaket wurden folgende Architekturen und Patterns verwendet:   * + MVC   + CQRS (Command Query Responsibility Segregation)   + CCD (Clean Code Development)   + DDD (Domain-driven Design)   + Repository   + Unit of Work |

|  |  |
| --- | --- |
| Vorgehensweise | |
| Zu Beginn wurden mögliche Technologien zur Umsetzung geprüft. Aufgrund der Vorkenntnisse des Entwicklers mit .NET-Technologie sowie der Programmiersprache C#, wurde ASP .NET Core MVC analysiert und ausgewählt. Da das Persistieren der Daten eine notwendige Funktionalität darstellte, wurde auch Entity Framework Core analysiert und schlussendlich akzeptiert.  Da sowohl Benutzerinteraktionen über ein Frontend als auch Schnittstellen für Sensoren und Alexa implementiert werden sollten, wurde das MVC-Pattern ausgewählt, wodurch die Entwicklung von interaktiven Views und RESTful APIs ohne großen Aufwand möglich ist. Zusätzliche konnten dadurch Funktionalitäten wiederverwendet werden. Für die Kommunikation zwischen den einzelnen Controllern und der Persistenz-Schicht fiel die Entscheidung auf das CQRS-Pattern (Command Query Responsibility Segregation).  Für das Persistieren der Daten entschied ich mich zur Verwendung des Repository Pattern und einer Unit of Work Implementierung. Clean Code Development (CCD) und Domain Driven Design sollten mir als Grundprinzipien dienen.  Nachdem die Planungsphase abgeschlossen war, wurde Visual Studio 2017 (in Folge VS) installiert. Entscheidende Workloads für die Umsetzung waren „ASP.NET and web development“ sowie „Data storage and processing“. Zusätzlich wurde das .Net Core SDK und zur Unterstützung der Entwicklungstätigkeit ReSharper von JetBrains installiert.  Es folgte Erstellung der Solution und des ESE.SmartHome Projekts in VS unter Auswahl des ASP.NET Core Web Application Templates. Als Target wurde jedoch das .NET Framework 4.6.2 eingestellt, weil ich zu Beginn noch keinen unmittelbaren Nutzen in einem reinen .Net Core Projekt sah. Mittels Nuget Packet Manager wurden in VS Packages von Microsoft, JetBrains und Serilog für das Logging installiert.  FeatureFolders von OdeToCode wurden ebenfalls installiert, um eine Strukturierung des Projekts abseits der üblichen MVC-Aufteilung zu ermöglichen. Dadurch konnten die einzelnen MVC-Teile Feature-bezogen in einem App-Ordner untergebracht werden. Das führte allerdings zu einem Problem mit ReSharper, der durch die neue Aufteilung nicht mehr voll funktionsfähig war. Die Lösung war die Erstellung einer Datei mit Assembly-Attributen, welche ReSharper die Locations der MVC-Komponenten miteilt.  Nachdem ich mich für das CQRS Pattern entschieden hatte, analysierte und installierte ich die MediatR Library, die es ermöglicht Commands und Queries sauber voneinander zu trennen. Dank einer Extension konnte MediatR rasch mittels Dependency Injection hinzugefügt werden.  Zum Erstellen und Testen der APIs sah ich mir Swashbuckle’s Swagger an, der einen großartigen ApiExplorer anbietet und bereits Core-tauglich ist. Nach der Installation wurde ein API TestController erstellt und mit Mocks versorgt. Das Ergebnis war überzeugend.  Damit nebenbei Alexa geplant und umgesetzt werden konnte, implementierte ich eine erste Version eines Alexa API Controllers. Mittels Swagger wurde ein YAML-Dokument generiert und ausgetauscht. Einfacher geht es wohl kaum.  Der nächste Schritt betraf den Data Access Layer. Ein zweites Projekt namens ESE.SmartHome.Core wurde in der Solution erstellt. Aufgrund der ausführlichen Dokumentation von Entity Framework Core (in Folge EF) installierte ich in dem neuen Projekt die notwendigen Nuget Packages. Beim Hinzufügen der Projektreferenz in das Hauptprojekt kam es leider zu Fehlern aufgrund des Target Frameworks. Ich entschied mich eine Migration des Hauptprojekts auf Core 2.0 durchzuführen. Sobald man herausgefunden hat, dass es hauptsächlich das Ändern der csproj-Datei erfordert, ist der Rest rasch erledigt.  Das DAL-Projekt konnte nun erfolgreich referenziert werden. Ich erstelle das Datenmodell in Form von Entities und fügte eine Abstraktion des DbContext hinzu. Die einzelnen DbSets wurden sofern notwendig mit einer Configuration versehen, welche für entsprechende Navigation und Beziehung der einzelnen Entities untereinander Sorge trägt. Beispielsweise werden so One-to-Many Beziehungen angegeben.  Repositories wurden erstellt, um Methoden auf den Entitites zu ermöglichen, wie beispielsweise simple CRUD Operationen. Zusammengefasst und bereitgestellt habe ich die Repositories in einer Unit of Work Klasse.  Controller und Klassen für Geräte und eine erste simple Auswertung wurden implementiert. Obwohl ich mit Problemen gerechnet hatte, operierten MediatR, UnitOfWork und MVC problemlos miteinander.  Wir wollten Echtzeitdaten anzeigen, also begann ich meine Recherche zu diesem Thema. SignalR wurde in vielen Tutorials und Beiträgen aufgelistet. SignalR ermöglicht es dem Server mittels Websockets Daten an die Clients zu pushen und funktioniert mit ASP .NET Core. Eine Beispielimplementierung aus dem Internet ermöglichte eine rasche Implementierung. Den Frontend-Teil zum Thema Echtzeitdaten werde ich im Arbeitspaket Frontend näher erläutern.  Als nächstes implementierte ich die RESTful API Endpunkte für die Sensoren. Da Swagger bereits implementiert war, ging das sehr zügig voran und kleine Bugs konnten schnell gefixt werden. Damit ich sowohl API als auch Echtzeitdaten testen konnte, erstelle ich ein drittes Projekt namens SensorMock. Dabei handelt es sich um eine simple Konsolenapplikation, die mehr oder minder zufällig erstellte Messdaten an den API-Endpunkt schickt. Bugfixes betrafen vorwiegend die Anzeige und somit den Frontend-Bereich.  Mitten in der Umsetzungsphase stellte sich heraus, dass Alexa für den Betrieb Zugriff auf die Web App über das Internet benötigte. Ich recherchierte dann Möglichkeiten für Deployment und Betrieb der Applikation auf einem erreichbaren Webserver. Aufgrund der bereits bestehenden, starken Microsoft-Zentrierung, analysierte ich deren Cloud Dienste, namentlich Azure App Services.  Ich erstellte mir einen Free Account und richtete diesen ein. Ein ausführliches Tutorial von Microsoft über das Publizieren von ASP .NET Core Web Apps auf Azure direkt über VS erleichterte den Deployment-Prozess. Ein kleiner Fehler in der Anleitung kostete mich schließlich doch ein wenig Zeit. Die Erstellung eines SQL Datenbank Servers funktionierte nicht in VS, sondern erst direkt im Azure Portal, weil bestimmte Regionen diesen Dienst nicht zulassen. Die Fehlermeldung dazu war allerdings nur im Azure Portal zu sehen und nicht in VS.  Das Publish-Profil war eingerichtet, und die Web App befand sich tatsächlich im Netz. Zu sehen war aber leider nur eine Error-Seite. Der Grund war, dass die Datenbank zwar vorhanden, aber leer war. Das ließ sich beheben durch eine „Database.Migrate“-Anweisung in der Startup-Datei. Absofort wurden Operationen zur Generierung von Schema-Änderungen automatisch von EF an Azure weitergegeben.  Erfolgreich getestet habe ich die Datenbank mit dem SQL Operations Studio, das eine einfache, rasche Anbindung an eine von Azure bereitgestellte Datenbank ermöglicht. Es müssen beim erstmaligen Verbinden lediglich Firewall Rules eingestellt werden. Dies kann im Azure Portal manuell, oder mittels SQL Operations Studio automatisch erledigt werden.  Die Arbeit an Alexa und den Sensoren war ebenfalls vorangeschritten, weshalb Anpassungen an den jeweiligen APIs notwendig wurden. Diese Phase nutzte ich für ein generelles Code Refactoring, überarbeitete den Data Access Layer, ermöglichte asynchronen Zugriff auf einige Repository-Methoden und kümmerte mich an einigen Stellen um ein besseres Exception Handling.  Nach der ersten erfolgreichen Kommunikation zwischen einem Sensor und der Web App, schränkte ich das Persistieren der Daten ein. Da es sich bei dem Azure Account um eine kostenlose Developer-Variante handelt, ist der Speicherplatz begrenzt. Messdaten wurden nun pro Gerät einmalig in die Datenbank geschrieben und in Folge lediglich aktualisiert.  Letzte Änderungen am Backend waren gefixte Bugs in der API, welche erst durch die abschließenden System-Tests zu Tage traten. |

|  |  |
| --- | --- |
| Lessons learned | |
| Meine Top 3 Lessons Learned   1. Code First zur Erstellung von Datenbanken ist an sich eine großartige Angelegenheit, aber ein wenig mehr Planung hätte mir bei der Datenmodellierung durchaus einiges an Refactoring erspart. Da ich eine relationale und somit nicht Schema-agnostische Datenbank eingesetzt habe, waren manche Adaptierungen mit Datenverlusten verbunden. EF Core leistet zwar gute Arbeit mit den Migrations, aber es kann nunmal eben nicht zaubern. 2. CQRS for the win. Ich habe zum ersten Mal dieses Pattern eingesetzt, und obwohl es für diese Solution eventuell nicht ideal ist, so hat es mir doch die Erstellung der einzelnen Features enorm erleichtert. Anstatt ein Viewmodel zu missbrauchen, weil man es in mehreren Views benötigt und somit immer mehr Properties hinzufügt, bis man eine eierlegende Wollmilchsau auf Krücken geschaffen hat, wird für jede Funktionalität eine spezialisierte Klasse implementiert. Eventuell verletzt man dabei das eine oder andere Mal das DRY (Don’t repeat yourself) Prinzip, aber dafür erreicht man ein sehr hohes Maß an Flexibilität. 3. Swagger ist großartig! Ich kann es nur empfehlen. Die schnelle Entwicklung mit Swagger hat mich sehr erstaunt. Mit ein paar wenigen Kommentaren und Attributen erhält man eine fertige, testbare API zur Verfügung gestellt. Auf Knopfdruck kann das zugehörige JSON-File heruntergeladen werden und in ein YAML transformiert werden. Ohne Swagger hätte das Team und ich mit Sicherheit mehr Zeit für die Entwicklung benötigt. |

|  |  |
| --- | --- |
| AP: Alexa Skill | |
|  | Verantwortlicher: Salko Nuhanovic Der folgende Abschnitt befasst sich mit der Implementierung des Alexa sowie der Aufsetzung des entsprechenden Environments.  „Alexa frage **bestes Projekt** nach **Temperatur im Abstellraum**.“  Jeder Alexa Skill hat einen sogenannten Invocation Name, welcher sozusagen der Name der App ist. In unserem Fall ist unser Invocation Name „Bestes Projekt“. Jede App kann mehrere Intents haben. Intents sind dann sozusagen die Features der App, im obigen Beispiel haben wir den Intent „Temperatur im Abstellraum“. Somit wird das Feature Temperatur von Bestes Projekt aufgerufen.  Die Spracherkennung wird in der Cloud durchgeführt, wo auch auf Basis der Triggerwords die entsprechenden Funktionen durchgeführt werden.  Grundlegend gibt es verschiedene Wege einen Alexa Skill zu schreiben.  Ein Alexa Skill besteht aus drei grundlegenden Komponenten:   * skill.json * Interaction Model * Business Logik   Im skill.json File wird der Skill anhand von Metainformationen (zb. Name, Kategorie ...) grundlegend beschreiben. Wichtig ist hier die Angabe, wo sich die Business Logik befindet. Prinzipiell gibt es zwei Möglichkeiten die Business Logik zu hosten. Entweder über AWS Lambda oder als Webservice. Wir haben uns aus Komfortgründen für die erste Variante entschieden.  Das Interaction Model (IM) beschreibt die Schnittstelle zum User, pro Sprache des Skills gibt es jeweils ein IM. Das IM setzt sich aus Intents und Slots zusammen.  Slots sind Parameter, die ein User einem Intent mitgeben kann um die Business Logik durchzuführen. In unserem Beispiel haben wir den Ort (zb.Abstellraum) der Wetterstation als Slot.  Je nach gewähltem Intent kann eine andere Business Logik durchgeführt werden.  Das Interaction Model wir anhand eines JSON Files beschrieben. Die Alexa Developer Console ermöglicht es das IM auch grafisch zusammenzustellen.  Hier ein Auszug des Interaction Models von „bestes projekt“:  {  "interactionModel": {  "languageModel": {  "invocationName": "bestes projekt",  "intents": [  {  "name": "status",  "slots": [  {  "name": "deviceID",  "type": "AMAZON.Room",  "samples": [  "Abstellraum",  "Kühlraum",  "Kühlschrank",  "Badezimmer",  "Küche",  "Wohnzimmer",  "Schlafzimmer"  ]  }  ],  "samples": [  "{deviceID}",  "Temperatur im {deviceID}",  "nach {deviceID}",  "erzähle mir etwas über {deviceID}",  "Information über {deviceID}",  "Status im {deviceID}"  ]  }  ]  },    }  }  Da entschieden wurde das AWS Lambda genutzt wird, musste entschieden erden in welcher Programmiersprache die Funktion geschrieben werden soll. AWS Lambda unterstützt folgende Sprachen:   * Java * nodeJS * Python * C# * Go   Um einen Einblick in die Welt von JavaScript als Backendsprache zu bekommen, haben wir uns für nodeJS entschieden.  Jede Lambda Funktion benötigt einen Handler welcher ausprogrammiert wird.  Unsere Lambda Funktion ruft die REST API des Backends in Azure auf und retourniert anschließend den Text welcher von Alexa vorgelesen wird.  Auszug aus Lambda Funktion (Aufruf der REST API):  var req = http.request(options, function (res) {  var chunks = [];  res.on("data", function (chunk) {  chunks.push(chunk);  });  //Verarbeite Response  res.on("end", function () {  var body = Buffer.concat(chunks);  var data = JSON.parse(body); //Parse JSON Objekt  var temperatur = data.temperature + '';  temperatur = temperatur.replace('.',',');  var text = 'Vielen Dank, dass du das beste Projekt nutzt. Die Temperatur in deinem ' + event.request.intent.slots.deviceID.value+ ' beträgt ' + temperatur + ' Grad Celsius' + ', und die Luftfeuchtigkeit beträgt ' + data.humidity + ' Prozent';  output(text,context); // Übergebe Antwort  });  });  Mittels event.request.intent.slots.deviceID.value wird es ermöglicht auf den übergebenen Slot (Parameter) zuzugreifen. Der Slot entspricht in unserem Fall, dem Raum für welchen die Messungen abgefragt werden  **Vorgehensweise**  Zu Beginn habe ich die Alexa API und dem Umfeld (AWS, AWS Lambda) studiert.  Nach eingehender Recherche und Durchsicht einiger How-To Artikeln, habe ich die ASK Command Line Interface (CLI) gefunden.  Die ASK CLI ist ein Tool um Alexa Skills und damit verbundene Lambda Funktionen zu managen. Die ASK CLI hat Zugang zur Skill Management API, welche es erlaubt Alexa Skills via Kommandozeile zu programmieren.  Die Challenge hinter der ganzen Programmierung des Skills, war das korrekte aufsetzen des AWS Accounts bzw. Alexa Developer Accounts.  Hier eine kurze Anleitung dazu:  Installation ASK CLI:  npm install -g ask-cli  Folgender Command initialisiert die ASK CLI mit den Amazon Developer account:  ask init    Nach Auswahl des Profils geht ein Browser Fenster auf bei welchem man sich bei Amazon einloggt:    Abbildung 12. Login ASK init  Nach erfolgreichem Login, werden Public Keys ausgetauscht was es erlaubt Konfiguration via ASK CLI am Developer Konto durchzuführen.  Nach erfolgreicher Einrichtung des Developer Accounts, ist noch die Konfiguration von Lambda erforderlich, dazu kann ich folgendes Guide empfehlen:  <https://developer.amazon.com/docs/custom-skills/host-a-custom-skill-as-an-aws-lambda-function.html>  Wichtig hierbei ist die Einstellung der richtigen Policy der Lambda Funktion, welche erlaubt das die Funktion öffentlich aufgerufen werden kann.  Nach erfolgreichem Setup der ASK CLI, des AWS Accounts (inkl. AWS Lambda) und des Amazon Developer Accounts kann mit der Implementierung gestartet werden.  Ich habe damit gestartet mit einen Hello-World Skill zu erstellen welchen ich dann weiterentwickelt habe.  ask new  Mit diesem Befehl erstellt ASK einen Hello-World Alexa Skill, mit folgender Struktur:    Abbildung 13 ask new  Wie oben beschrieben werden die Hauptbestandteile Business Logik(lambda), Interaction Model (models) und das skil.json File erstellt.  Um den Alexa Skill zu testen gibt es zwei Wege:  Alexa fähiges Gerät (welches mit entsprechendem Amazon Konto verbunden ist)  Test Feature der Amazon Developer Konsole  Bevor der Skill getestet werden kann muss er mittels des deployed werden:  ask deploy  Nach erfolgreichem Deployment ist der Skill in der Developer Console ersichtlich:    Abbildung 14 Amazon Developer Konsole  **Intent**  Im linken Navigationsbereich kann das Interaction Model bearbeitet werden. Unser Projekt hat folgendes Interaction Model:    Abbildung 15 Intent Temperatur  Unser Projekt stellt einen Intent zur Verfügung welcher Messdaten des übergebenen Slots {deviceID} abfragt.  Sample Utterances sind Triggerwords auf welche Alexa hört um den Intent zu triggern. Sprich wenn Alexa nach dem Invovking word („bestes projekt“) einer dieser Phrasen hört, wird der Intent durchgeführt und der Slot an die Lambda Funktion übergeben.  Beispielsweise wir wollen den Temperatur Intent ausführen mit dem Abstellraum als Slot, dann können wir sagen:  „Alexa öffne bestes Projekt und erzählt mir etwas über Abstellraum“ Lambda Funktion Unter der Kategorie Endpoint wird der Link der Lambda Funktion eingetragen welcher vom Alexa Skill aufgerufen werden soll.    Abbildung 16 Lambda Funktion Testen Prinzipiell ruft Alexa mit einem JSON Request via HTTP die Lambda Funktion auf.  Dies kann direkt in der Developer Konsole getestet werden:    Abbildung 17 Testing Alexa Skill in Developer Konsole  Im JSON Input ist zu erkennen das der Slot deviceID mit dem Wert Küche an als Parameter beim Aufrufen der Lambda Funktion übergeben wird.  Die Lambda Funktion senden dann eine Antwort in Form eines JSON Objekts zurück, welches die Output Speech für Alexa enthält. Lessons Learned  * „You can do everything with JavaScript“ 🡪 nodeJS eignet sich als Backendsprache sehr gut * Interaction Model muss gut durchdacht werde, anfangs hatten wir große Probleme das Alexa nicht ausgeführt hat bzw. einen andere Skill durchgeführt hat, dazu kommt das Alexa keinen Dialekt versteht z.B. wird „frag“ nicht verstanden 🡪 „frage“ wäre das richtige Linkingword um Invoking mit Intent zu verbinden * Postman hat einen netten Code Generator (https://www.getpostman.com/) * „Never Ever use your AMEX for AWS“ 🡪 da ich meine AMEC Kreditkarte als Zahlungsmittel bei Amazon Web Services hinterlegt habe, und AMEX Amazon nicht abbuchen lassen konnte, weil ich es nicht bestätigt habe, wurde mein AWS Konto als auch mein normales Amazon Konto gesperrt. Um dies zu lösen musste ich eine Stunde lang mit der AWS Hotline telefonieren. * ASK CLI 🡪 gutes Tool zum Verwalten vom Alexa Skills * AWS bietet direkt ein Service an um Big Data welche von IoT Devices kommen (z.B. unserer Wetterstation) via Amazon Kinesis zu analyiseren  Further Work Das Fehlerhandling der Lambda Funktion kann noch erweitert werden, zB. wenn der User im Slot einen Raum übergibt welcher nicht registriert ist, kann Alexa eine schöne Fehlermeldung geben in welcher eine Anleitung beschrieben ist wie man ein Gerät richtig registriert |
|  | Verwendete Tools:  * Visual Studio Code (<https://code.visualstudio.com/>) * nodeJS (<https://nodejs.org/en/>) * ASK CLI (<https://www.npmjs.com/package/ask-cli>) * Amazon Developer Console (<https://developer.amazon.com/alexa>) * AWS Lambda (<https://developer.amazon.com/docs/custom-skills/host-a-custom-skill-as-an-aws-lambda-function.html>) * Postman (<https://www.getpostman.com/>) |
|  |  |

|  |  |
| --- | --- |
| AP: Frontend Implementierung | |
|  | Verantwortlicher: Stephan Nöhrer Das Frontend stellt die Schnittstelle zwischen dem Backend und den Benutzern der Web App dar. Wie im Arbeitspaket Backend beschrieben, wird die Trennung logisch in Client/Server vorgenommen. Das Frontend nutzt nach MVC-Pattern die View-Models, um mit dem Backend zu kommunizieren.  Benutzer können Devices (die Sensoren) erfassen und dabei eine MAC Adresse angeben, wodurch die Devices bei der Inbetriebnahme ihre eindeutige ID erhalten, welche zum Senden der Messdaten benötigt wird. Ein angegebener Name dient Alexa zur Abfrage der Messdaten des jeweiligen Devices.  Die Angaben zu den Devices können bearbeitet werden und bei Bedarf lassen sich Devices auch deaktivieren beziehungsweise reaktivieren. Deaktivierte Devices werden vor Alexa verborgen. Außerdem können angelegte Devices gelöscht werden.  Zusätzlich bietet das Frontend die Anzeige von Echtzeitdaten und historischen Daten an, wobei die historischen Daten im Prototyp nur eingeschränkt zur Verfügung stehen, um die Datenbank in der Azure Cloud nicht übermäßig zu beanspruchen. |

|  |  |
| --- | --- |
| Verwendete Tools | |
| Für das Arbeitspaket wurden folgende Tools verwendet:   * + Visual Studio 2017 (d)   Version: 15.7.4  IDE von Microsoft   * + SQL Operations Studio (e)   Version: 0.24.1  Zum Verwalten von Azure SQL Databases |
| JavaScript, CSS und Tools | |
| Für das Arbeitspaket wurde Folgendes verwendet:   * + [Gulp](https://gulpjs.com/)   + [Bootstrap](https://getbootstrap.com/)   + [jQuery](https://jquery.com/)   + [Font Awesome](https://fontawesome.com/)   + [Parallax.js](http://pixelcog.github.io/parallax.js/)   + [Popper.js](https://popper.js.org/)   + [Bootstrap Toogle](http://www.bootstraptoggle.com/) |

|  |  |
| --- | --- |
| Vorgehensweise | |
| Die Planung für das Frontend begann zeitgleich mit der Planung des Backend-Bereichs. Die Implementierung konnte jedoch erst ihren Anfang nehmen, nachdem die grundsätzliche Infrastruktur im Backend abgebildet war.  Nach Erstellung des ersten Controllers und dessen Model wurde eine Index-Seite im Razor View Format generiert und als Startseite festgelegt. Das diente lediglich dazu, das Zusammenspiel zwischen den einzelnen MVC-Komponenten zu testen.  Als nächstes wurde das Master-Template „\_Layout“ erstellt, welches von den meisten Views, bis auf einige Partial Views, verwendet wird. Sinn und Zweck war das Bereitstellen eines durchgängigen Designs sowie dem Einbinden der JavaScript- und Stylesheet-Dateien an zentraler Stelle.  Die meisten JavaScript-Libraries und CSS-Dateien standen bereits fest, sollten jedoch auf eine möglichst simple Art und Weise hinzugefügt, verändert oder entfernt werden können. Dazu sollte ein spezielles Build-Tool verwendet werden. Ich recherchierte einige dieser Tools, entschied mich jedoch nach kurzer Analyse für Gulp.  Gulp sowie eine Vielzahl an Node Modules wird bei der Installation von Visual Studio mit dem Web Development Workload automatisch installiert. Damit es in dem Projekt verwendet werden kann, müssen in einer package,json Datei lediglich die gewünschten Dependencies eingetragen und eine gulpfile.js Datei erstellt werden. In der gulpfile.js Datei werden die gewünschten Tasks in JavaScript geschrieben. Zu installierende Node Modules werden in einer assetsconfig.json Datei eingetragen. Mithilfe eines Watch-Tasks werden Änderungen in der assetsconfig.json Datei automatisch von Gulp erkannt und in das wwwroot-Verzeichnis übernommen.  jQuery, Bootstrap und andere benötigte Pakete wurden in der assetsconfig.json Datei eingetragen und in dem Master-Template „\_Layout“ als benötigte Referenzen eingetragen. In Folge wurde von mir das Navigations-Menu geschrieben. Das erste Menu war als Sidebar gestaltet. Während der voranschreitenden Entwicklung der einzelnen Views stellte sich jedoch heraus, dass dies in keiner Weise responsive war. Deswegen wurde gegen Ende des Projekts das Menu auf eine Top-Bar umgestellt, wodurch eine Nutzung der Web App mittels Mobile Device ermöglich wurde.  Ein großes Thema war die Darstellung von Echtzeitdaten. Das Backend stellte bereits einen SignalR für serverseitiges Pushen von Daten zur Verfügung. Ich begann die Recherche nach einer Library zur Anzeige von Charts. Relativ rasch wurde ich auf Chart.js aufmerksam und fand sogar ein Beispiel, das SignalR verwendete. Die Installation war dank Gulp eine Kleinigkeit und es folgten einige spielerische Experimente.  Grundsätzlich funktionierten sowohl Datenübernahme als auch Anzeige, jedoch gab es Probleme bei der Umsetzung von mehreren Werten und die Gestaltung der Achsen sowie einer Legende war eine nervenaufreibende Angelegenheit.  Nach dem Drehen einiger Schrauben und dem Lesen unzähliger Stackoverflow-Posts, funktionierte es wie gewünscht. Es wurden nun die restlichen Views im Zusammenspiel mit dem Backend implementiert. Das Erfassen, Bearbeiten und Löschen von Devices war rasch umgesetzt, bis auf einen Bug beim Bearbeiten.  Beim Bearbeiten wird ein Modal geöffnet, was an sich kein Problem darstellt, jedoch bildete ich mir ein, dass die Aktivierung beziehungsweise Deaktivierung eines Devices mittels Toggle Switch zu geschehen hat. Die View zum Bearbeiten wird als Partial View von einem Ajax Call generiert. Beim erstmaligen Öffnen des Modal‘s, funktionierte es wie vorgesehen, aber jeder zweite Aufruf sorgte dafür, dass Funktionalität und Style des Switches nicht mehr vorhanden waren. Es hat mich Stunden gekostet dahinter zu kommen. Das Problem war im Nachhinein betrachtet mehr als logisch, aber nach Stunden der Entwicklung fehlte mir offenbar der Blick dafür.  Durch den Ajax Call verlor der Toggle seine Eigenschaften. Beim ersten Mal funktionierte es, weil die benötigten Referenzen durch den Server gegeben waren, aber danach clientseitig nicht mehr. Die Lösung war eine Erweiterung des Ajax Abschnitts. Der Toggle Switch wurde nun nach Erstellung des Modal‘s clientseitig zerstört, und dann erneut generiert. Leider musste dadurch auch der Style extra ergänzt werden, siehe Abbildung 9.    Abbildung 18 Lösung für Bootstrap Toggle Problem  In der letzten Phase wurde wie bereits erwähnt die Darstellung beinahe vollständig verändert. Aufgrund neu gewonnener Erkenntnisse während eines Projekts für die Lehrveranstaltung Information Engineering, wollte ich das Aussehen „verbessern“. Das Menu wurde mittels Parallax.js um ein Scroll-bares Image erweitert. Ebenso kam ein Footer hinzu und generell wurde Wert auf Responsiveness gelegt. Abgesehen vom Navigations-Menu erfordert dies jedoch kaum große Änderungen, dank Bootstrap.  Während den abschließenden System-Tests wurde eine View zur Anzeige von Fehlern hinzugefügt und die Echtzeitdaten-View um ein Drop-Down Menu zur Auswahl eines Devices erweitert, um den Umgang mit mehreren Devices zu ermöglichen. |

|  |  |
| --- | --- |
| Lessons learned | |
| Meine Top 3 Lessons Learned   1. Frontend nimmt viel Zeit in Anspruch. Viel mehr Zeit als ich erwartet hatte. Cascading Style Sheets sind eine Geschichte für sich. Jede kleine Änderung mündet eventuell in einer Katastrophe. Und Katastrophen gab es hierbei einige. Ein Tool, das ich während meiner Arbeit sehr zu schätzen gelernt habe, ist Chrome DevTools. – Danke Google. 2. AJAX ist fein, bis es irgendwann nicht mehr fein ist. Bei der Implementierung von Bootstrap Toogle und einem über AJAX erstellten Modal-Fenster, wurde viel Zeit verbraten, bis ich den dynamischen Aspekt clientseitiger Entwicklung verstanden habe. Hier hat mir vor allem Stackoverflow sehr geholfen. 3. Es kommt immer anders als man denkt beziehungsweise agil bleiben. Der ursprüngliche UI-Plan konnte von mir aufgrund mangelnder Erfahrung nicht 1:1 umgesetzt werden. Es ist zwar wichtig nicht sofort aufzugeben, aber ein Zeitlimit sollte man sich auf jeden Fall setzen. Wenn es nicht funktioniert wie man möchte, weil man eventuell Vorstellungen hatte, die nicht zu erfüllen sind, sollte man sich nach eingehender Analyse und besserem Verständnis für die Materie eingestehen, dass es eben genau so umsetzbar ist. Dann hilft es einen Moment Pause zu machen und im Anschluss einen neuen Weg zu beschreiten. |

|  |  |
| --- | --- |
| AP: Embedded Integration | |
|  | Verantwortlicher: Hannes Aurednik Der folgende Abschnitt befasst sich mit der Integration der einzelnen Embedded-Module in eine gesamtheitliche Lösung.  Anfangs werden verschiedene Integrationsszenarien verglichen und die Vorzüge der gewählten Variante dargelegt.  Im Zuge des Arbeitspaketes mussten daher einige Änderungen an den bestehenden Modulen durchgeführt werden. Auf diese wird in weiterer Folge näher eingegangen.  Zusätzlich wurde das Arbeitspaket verwendet um Refactoring-Maßnahmen durchzuführen die den Programmcode selbst oder seine Lesbarkeit optimieren. |

|  |  |
| --- | --- |
| Analyse und Bewertung Integrationsszenarien | |
|  | Zunächst Analyse und Bewertung möglicher Integrationsszenarien:   * Mutexes & Semaphore * Queue * Mailboxen  Mutexes & Semaphore Bei Mutexes handelt es sich um binäre Semaphore die benutzt werden um simplen wechselseitigen Ausschluss zu gewährleisten. [3]  Obwohl Mutexes in Kombination mit globalen Variablen für die Kommunikation zwischen dem Sensorik-Task und dem Netzwerk-Task verwendbar wären besitzen sie einige Nachteile weswegen von einer Verwendung abgesehen wurde. [4]  Erstens muss sichergestellt werden, dass ein Task der den Token besitzt diesen auch wieder zurückgibt damit ein anderer auf die Ressource zugreifen kann.  Zweitens besteht zwar die Möglichkeit eine Sperrzeit zu spezifizieren welche in Ticks angegeben wird, jedoch besteht kaum Möglichkeit die Laufzeit eines HTTP Requests im Voraus zu erahnen.  Drittens erfordert die Verwendung eine engere Verzahnung der einzelnen Softwaremodule statt größtmögliche Entkoppelung zu gewährleisten. [4] Queue Queues haben verfügen über ähnliche Eigenschaften wie Mailboxen. [5]  Der Unterschied besteht primär darin, dass Queues auf doppelt verlinkten Listen und Mailboxen auf Kopien basieren.  Queues besitzen aber keine definierte Maximalgröße. [6]  Da durch vorhergehende Lehrveranstaltungen aber bereits Erfahrungen mit der Verwendung von Mailboxen bestehen wurde diese Variante präferiert. Mailboxen Zusätzlich zu den bestehenden Vorerfahrungen des Projektteams mit Mailboxen bieten diese einen weiteren Vorteil. Mit ihnen ist es möglich sicherzustellen, dass der Input an Nachrichten nicht die Anzahl der bearbeitbaren übersteigt. [7]  Die gewählte Lösung mittels Mailboxen wird im nächsten Abschnitt genauer erläutert. |

|  |  |
| --- | --- |
| Integration mittels Mailboxen | |
|  | Um eine möglichst hohe Entkoppelung der Softwaremodule zu erreichen und dennoch eine stabile und performante Lösung zu erhalten wurden für die Kommunikation zwischen den Tasks Mailboxen eingesetzt.  Diese bieten zusätzlich die Möglichkeit auf einfache Art weitere Sensoren und entsprechende Tasks hinzuzufügen ohne hohen Änderungsaufwand bei anderen Modulen zu erzeugen. |

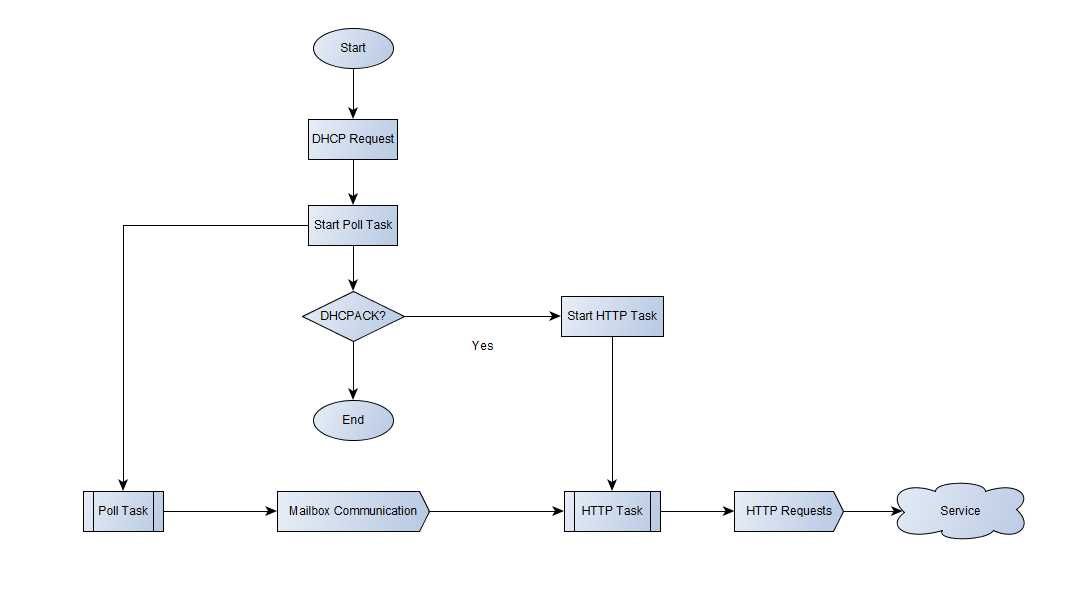


Abbildung 19: Ablaufdiagramm Embedded Lösung

|  |  |
| --- | --- |
|  | Die obige Grafik zeigt den schematischen Ablauf der implementieren Lösung.  Nach dem Start des Mikrocontrollers wird zunächst ein DHCP Request ausgesendet um eine IP-Adresse zu erhalten und die Sensorik Funktionalität gestartet.  Anschließend wartet das Gerät bis es eine IP Adresse erhält. Sollte dies nicht der Fall sein wird die Applikation beendet da es keinen Sinn macht Werte zu ermitteln für die kein Konsument existiert bzw. davon ausgegangen werden muss, dass ein Fehler im Netzwerk vorliegt.  Sobald das Gerät eine gültige Adresse erhält wird der http Task gestartet.  Hat dieser die, in Abschnitt „5 AP Netzwerk“ nötigen Schritte näher beschriebenen Schritte, zum Aufbau einer Serververbindung vollführt werden die Messwerte per Mailbox empfangen und an den Server weitergeleitet.  Die Größe der Mailboxen ist beschränkt und stellt somit sicher, dass es zu keinem Überlauf kommt wenn der http Task nicht mehr in der Lage ist Daten zu versenden, der Sensorik Task aber weiterhin Werte sendet.  Um das Hinzufügen von neuen Funktionen zu erleichtern wurden für Temperatur und Luftfeuchtigkeit unterschiedliche Mailboxen implementiert.  Alternativ hätte auch mittels einem struct ein eigenes Nachrichtenformat definiert werden können. |

|  |  |
| --- | --- |
| Aufgetretene Herausforderungen | |
|  | In diesem Abschnitt werden die Herausforderungen/Probleme erläutert die im Zuge des Arbeitspaketes aufgetreten sind. Zusätzlich wird beschrieben wie diese gelöst wurden und welche Tools oder Recherchequellen dafür dienlich waren. Anpassung Systemkonfiguration Im Zuge der Integration musste die Systemkonfiguration angepasst werden, um den Formatspecifier %f zu unterstützen.  Dies ist in der Standardkonfiguration nicht der Fall, in diesem Setup aber für das Erstellen des JSON Strings essentiell. Anpassung SW-Architektur Aufgrund der Anforderungen an die Software mussten die recherchierten Beispielcodes so umgebaut werden, dass sie als Funktionen aus einem Task aufgerufen werden können. In der ursprünglichen Implementierung waren diese selbst als Task designed und hätten bei unveränderter Verwendung so unweigerlich zu Fehlern und ausbleibender Funktionalität geführt. |

|  |  |
| --- | --- |
| Refactoring-Maßnahmen | |
|  | Der Abschnitt erläutert die durchgeführten Refactoring-Maßnahmen sowie warum diese gesetzt wurden und welche Auswirkungen diese haben.  Folgende Refactoring-Maßnahmen wurden im Zuge des Arbeitspaketes durchgeführt:   * Entfernen doppelter Includes * Entfernen von überflüssigen Kommentaren * Entfernen von Hilfswerkzeugen der Entwicklung z.B. Hilfsfunktionen * Conditional Compilation  Entfernen doppelter Includes Dies ist eine Maßnahme um Fehler durch inkonsistente Definitionen zu vermeiden.  Zusätzlich wird der Zeitaufwand des Kompilierens reduziert wenn Schritte nicht mehrfach ausgeführt werden müssen. Entfernen überflüssiger Kommentare Nicht mehr benötigte Kommentare zu entfernen erleichtert einerseits die Lesbarkeit der Sourcefiles andererseits hilft es auch Verwirrungen zu reduzieren.  Oft handelt es sich um Artefakte des Debuggings welche in der fertigen Lösung keinen Zusammenhang besitzen. Entfernen von Hilfswerkzeugen der Entwicklung Das Entfernen von Hilfs-Konstrukten ist nur eine von vielen Maßnahmen um den eigenen Programmcode verständlicher und leichter lesbar zu machen.  Ein weiterer Benefit ist, dass Verwirrungen vermieden werden indem nicht mehr benutzte Codeteile entfernt wird. Ein späterer Leser ist also nicht mehr in Versuchung Verweisen zu folgen die keine Verwendung mehr finden und für die Funktionalität keine Relevanz besitzen. Conditional Compilation - stellt die Möglichkeit dar, dass ein Compiler, abhängig von definierten Parametern, unterschiedliche Programme erzeugt.  Dies hat den Nutzen, dass beispielsweise durch das Hinzufügen einer einzelnen Zeile „#define DEBUG“ das spätere Programm dem Benutzer viel mehr Zwischeninformationen der Verarbeitung anzeigt als üblich.  Dies erzeugt speziell für das Debugging von Software einen hohen Mehrwert.  Die Grafik unterhalb zeigt ein Beispiel wie ein solcher abhängiger Codeabschnitt aussehen kann.    Abbildung 20: Beispiel Conditional Compilation  Diese Ausgabe wird nur in den Kompilierungsprozess miteinbezogen und dadurch im späteren Programm ausgeführt wenn das Schlüsselwort „DEBUG“ definiert wurde. |

|  |  |
| --- | --- |
| Installationsanleitung | |
| Da es sich bei dieser Lösung um kein fertiges Produkt handelt sind für die Inbetriebnahme einige Schritte zu beachten.  Diese finden Sie unter:  <https://github.com/salkonuhannovic/ese-project/blob/master/README.md> |

# Anhang

## Verwendete Tools

(a) Webhook., *Tool zum Testen von diversen HTTP Requests*., <https://webhook.site>

(b) Apitester., *Tool zum Testen von APIs*., <https://apitester.com>

(c) CCS., *IDE zur Entwicklung von embedded Projekten*., Code Composer Studio

(d) Visual Studio 2017, IDE von Microsoft, <https://visualstudio.microsoft.com>

(e) SQL Operations Studio, Tool zum Verwalten von SQL Storages, [Link](https://docs.microsoft.com/en-us/sql/sql-operations-studio/what-is?view=sql-server-2017)

(f) yEd., *Tool zum Erstellen von Diagrammen*., <https://www.yworks.com/products/yed>?

(g) Microsoft Powerpoint

## Abbildungsverzeichnis

[Abbildung 1: Netzwerktopologie Abschlusspräsentation 21](#_Toc518321679)

[Abbildung 2: Mockup Hauptseite 24](#_Toc518321680)

[Abbildung 3: Mockup Echtzeitdaten 24](#_Toc518321681)

[Abbildung 4: Mockup Geräteauswahl 25](#_Toc518321682)

[Abbildung 5: Architektur 26](#_Toc518321683)

[Abbildung 6: Netzwerktopologie Betriebszenario 32](#_Toc518321684)

[Abbildung 7: Ablaufdiagramm http Task 33](#_Toc518321685)

[Abbildung 8: HTU21D Nullwerte bei Ausgabe mit Debugging Hilfen 39](#_Toc518321686)

[Abbildung 9: HTU21D Variablenwerte 39](#_Toc518321687)

[Abbildung 10: Debugging Hilfe 40](#_Toc518321688)

[Abbildung 11: Korrekte Variablenwerte 40](#_Toc518321689)

[Abbildung 12. Login ASK init 51](#_Toc518321690)

[Abbildung 13 ask new 52](#_Toc518321691)

[Abbildung 14 Amazon Developer Konsole 53](#_Toc518321692)

[Abbildung 15 Intent Temperatur 53](#_Toc518321693)

[Abbildung 16 Lambda Funktion 54](#_Toc518321694)

[Abbildung 17 Testing Alexa Skill in Developer Konsole 55](#_Toc518321695)

[Abbildung 18 Lösung für Bootstrap Toggle Problem 60](#_Toc518321696)

[Abbildung 19: Ablaufdiagramm Embedded Lösung 64](#_Toc518321697)

[Abbildung 20: Beispiel Conditional Compilation 67](#_Toc518321698)

ANMERKUNG: Dieses Abbildungsverzeichnis generiert sich selbst.

## Quellenverzeichnis

[1] Concat Function, <https://stackoverflow.com/questions/8465006/how-do-i-concatenate-two-strings-in-c>

[2] POST Function, <https://e2e.ti.com/support/embedded/tirtos/f/355/t/555614?HTTP-POST-sample>

[3] Binary Semaphores, <https://www.freertos.org/Embedded-RTOS-Binary-Semaphores.html>

[4] Mutexes, <https://www.freertos.org/Real-time-embedded-RTOS-mutexes.html>

[5] Queue, <https://e2e.ti.com/support/embedded/tirtos/f/355/t/281235?Sys-Bios-Queue>

[6] Queues S125, <http://www.ti.com/lit/ug/spruex3t/spruex3t.pdf>

[7] Mailboxes S124, <http://www.ti.com/lit/ug/spruex3t/spruex3t.pdf>