16:12 Montag 12. Mai — github.com — 32 %

main | Batteriestandskontrolle_Rasierer / Dokumentation / Requirements.md | ↑ Top

Preview | Code | Blame | Raw

1.1	Automatische Spannungsmessung	Die Spannung des Akkus wird zuverlässig gemessen.
1.2	Ladezustandsberechnung in Prozent	Die Spannung wird zuverlässig in Prozent umgerechnet.
2.1	Batteriestand wird beim Einschalten angezeigt	~~Während der Nutzung wird der~~ aktuelle Batteriestand ~~angezeigt~~. 1 Anzeige
2.2	Einfache Statusanzeige bei Knopfdruck	Bei kurzem Drücken des An-/Aus-Knopfes wird der aktuelle Batteriestand angezeigt.
2.3	Restlaufzeit auf Knopfdruck	Bei erneutem kurzem Drücken des An-/Aus-Knopfes wird die Restbetriebszeit angezeigt.
2.4	Anzeige bei Inaktivität	Die Batteriestandskontrolle funktioniert auch nach längerer Nichtnutzung zuverlässig.
2.5	Anzeige bei Interaktion aktualisieren	Die Anzeige wird bei jeder Benutzerinteraktion automatisch aktualisiert.
3.1	Spannungsreferenz definieren	Es werden feste Spannungswerte für leer und voll als Referenz zur Anzeige genutzt.
3.2	Initiale Kalibrierung	Beim ersten vollständigen Lade- und Entladezyklus erfolgt eine Kalibrierung.
3.3	Rekalibrierung nach Ladezyklen	Nach mehreren vollständigen Ladezyklen erfolgt automatisch eine neue Kalibrierung.
4.1	Warnung bei niedrigem Akkustand	Der Nutzer wird eindeutig gewarnt, wenn der Akkustand niedrig ist.
4.2	Warnschwelle einstellbar	Der Benutzer kann einstellen, ob die Warnung bei 30 % oder 10 % Restladung erfolgt.
4.3	Warnsignal bei Unterschreitung	Bei Unterschreiten der Warnschwelle gibt es ein visuelles ~~und akustisches Signal~~.
5.1	Ladeaktivität anzeigen	Der laufende Ladevorgang wird klar angezeigt.
5.2	Ladeabschluss anzeigen	Es wird eindeutig angezeigt, wenn der Akku vollständig geladen ist.
5.3	Ladefehler anzeigen	Der Nutzer wird visuell informiert, wenn der Ladevorgang fehlschlägt.
5.4	Ladezustand elektronisch erkennen	Die Ladeelektronik erkennt, ob der Ladevorgang aktiv, abgeschlossen oder fehlgeschlagen ist.
5.5	Rasur während Laden signalisiert	Während des Ladevorgangs gibt es eine visuelle Meldung, ~~wenn eine Rasur möglich ist.~~

Abhängigkeiten:

1.1 → 1.2

1.2 → 2 (Epic)

2.1 → 2.*

3.1, 3.2 → 2.1

5.1 – 5.3 → 5.6

1.1, 3.1 → 6.1, 6.2

6.* → 3.5

Zusammengehörigkeiten:

4.1, 4.2

62, 6.3, 5.*

6.1, 1.*

2.6, 4.4

Sprint 1: Erkennung + Anzeige → 1 & 2 (+ paar von 3)

Sprint 2: Laden + Überladung + Temp. → 5, Teile von 6

Sprint 3: Warnung + restliches (Feinheiten) → 4

16:12 Montag 12. Mai — github.com — 32 %

main | Batteriestandskontrolle_Rasierer / Dokumentation / Requirements.md | ↑ Top

Preview | Code | Blame | Raw

6.1	Unterspannungsschutz	Das Gerät schaltet sich bei kritischem Akkustand automatisch ab, um Tiefentladung zu vermeiden.
6.2	Überladeschutz	Der Ladevorgang wird automatisch bei Erreichen der maximalen Spannung abgebrochen.
6.3	Temperaturüberwachung beim Laden	Die Temperatur wird während des Ladevorgangs kontrolliert.

2. Nicht-funktionale Requirements

Nr.	Titel	Beschreibung
1.3	Keine zusätzliche Bedienung	Die Batteriestandsanzeige erfordert keine zusätzliche Interaktion zur Rasur.
2.6	Intuitive Anzeige	Der Batteriestand wird in einer für den Nutzer klar verständlichen Form dargestellt.
2.7	Keine störende Helligkeit	Die Anzeige ist nicht zu hell oder visuell aufdringlich.
2.8	Barrierefreie Anzeige (Farbenblindheit)	Die Anzeige ist auch für farbenblinde Nutzer eindeutig erkennbar.
2.9	Barrierefreie Anzeige (Sehschwäche)	Die Anzeige ist auch für Nutzer mit eingeschränktem Sehvermögen gut lesbar.
2.10	Anzeige ohne Verzögerung	Die Anzeige erfolgt innerhalb von <1 Sekunde nach Benutzerinteraktion.
3.4	Anzeigetoleranz	Die Prozentanzeige muss mit einer Toleranz von ±5 % gegenüber der realen Kapazität übereinstimmen.
3.5	Alterungsrobustheit	Auch bei <80 % Akkukapazität nach mehreren Jahren bleibt die Anzeige zuverlässig.
3.6	Ressourcenschonende Kalibrierung	Die Kalibrierung darf nur minimal Rechen- und Speicherressourcen beanspruchen.
4.4	Wahrnehmbarkeit der Warnung	Die Warnsignale müssen in typischen Alltagssituationen gut wahrnehmbar sein (z. B. im Bad oder bei Geräuschkulisse).
5.6	Ladeanzeige energieeffizient	Die Ladezustandsanzeige muss stromsparend sein und den Ladevorgang nicht negativ beeinflussen.

3. Abhängigkeiten & Konflikte zwischen Requirements

Tasks Sprint 1:

Recherche

- Rechercheaufgaben → Ergebnisse dokumentieren

- LED (rot blinken: low battery)
- überarbeiten: (rot leuchten: lädt)
  (grün leuchten: voll geladen)

Architektur

- Architekturmuster festlegen
- Komponentendiagramm erstellen
- Schnittstellendefinition erstellen
- Technologiestack festlegen

Design

- Klassendiagramme erstellen
- Sequenzdiagramme erstellen
- Zustandsdiagramme erstellen
- Möglichkeit Verwendung Design Patterns prüfen

Implementierung

- Klassen, Automaten, Operationen, Schnittstellen implementieren
- Traceability Matrix
- Code Metriken erstellen

Test

- Testfälle aufstellen, durchführen
- Code Review durchführen
- Testdoku

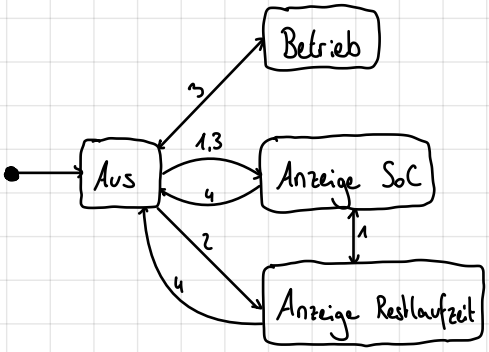
! Farbenblindheit in Sprint 2

Nachbereitung

- Fehlerverfolgung
- Retro (evtl. neu aufgetauchte Requirements)

Zustandsdiagramm Knopfinteraktion

nicht-deterministisch, endlich



1: Knopf 1x kurz gedrückt

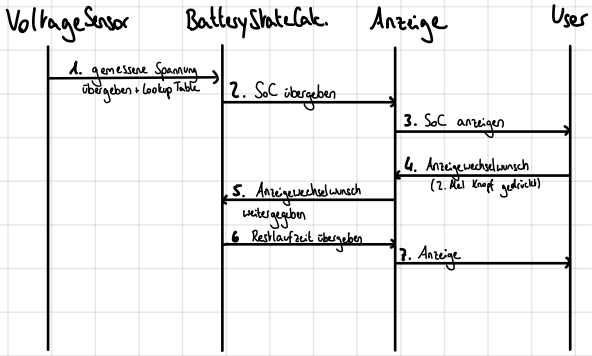
2. Knopf 2x kurz gedrückt

3. Knopf 1x lange gedrückt

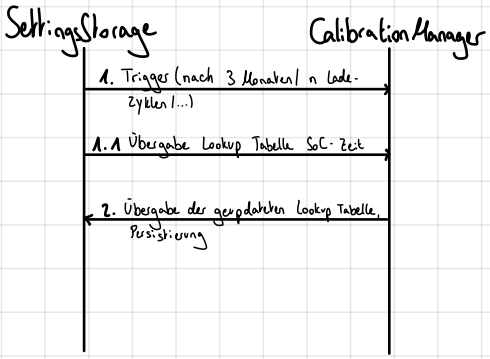
4. 5s kein Drücken

Klasse	Pattern	Grund
`InteractionHandler`	Command	Flexible Benutzerinteraktionen
`DisplayDriver`	Bridge	Entkopplung von Logik und Hardware
~~`BatteryStateCalculator`~~	~~Strategy~~	~~Austauschbare Berechnungsstrategien~~
`SettingsStorage`	Singleton (Optional)	Zentraler, konsistenter Zugriff auf gespeicherte Werte
`StatusDisplayController`	Facade	Einfaches Interface zur Anzeige

Sequenzdiagramm Strommessung zu Anzeige



Sequenzdiagramm Kalibrierung



User Interface:

Simple GUI → hat Button/Controller, erzeugt JFrame Anzeige

LEDPanel → steuert LED in GUI an

LEDController → bietet Methoden zum ansteuern des LED Panels an, enthält LEDPanel

VisOutputCont. → enthält LEDController
muss übergeben bekommen, welche Anzeige aktiv & ob Warnung

! keine Sound-Warnung

! Problem LED → Rot-Grün-Schwäche

PersistenceManager

CalibrationData → Hilfsklasse um Calib-Arrays gemeinsam handeln zu können

SettingsStorage → initiale Kalibrierung & Spannungsgrenzen drin, Warnschwelle & Volt → SoC in Dateien, auslesen wird public als API angeboten
(Singleton)

Hardware Abstraction

VoltageSensor → liest simulierte Spannungszahlen & stellt diese bereit

VoltageSimulator → Simulation statt HW, simuliert für 3 States (Laden, passive Entladung & Betrieb) die Spannungsverläufe nach Zeit
→ in ChargingState-Enum

ChargingState → Enum für Simulator (Laden, Entladen ug. Betrieb - Aktiv, Entladen ohne Betrieb - passiv)

ButtonInput → handelt ButtonInput (von JButton aus GUI) - erkennt short oder long Press und Inaktivität > 5s, ruft InteractionHandler auf bei: Klick

---

Battery Logic

BatterySateController → berechnet SoC aus aktueller Spannung (von Sensor), gibt zurück wenn LowBattery (liest Files aus)
Interpolation mit CalibData

~~CalibManager~~ - erstmal keine Rekalibrierung

InteractionManager → realisiert Zustandsdiagramm Knöpfe (ohne Restlaufzeit, aktuelle States können ausgelesen werden

OperationStates → an/aus (Betrieb)

DisplayStates → aus/an (Akkustand in %)

Tasks Sprint 2.

Recherche

- Rechercheaufgaben → Ergebnisse dokumentieren

Architektur

- Komponentendiagramm erstellen
- Schnittstellendefinition erstellen

Design

- Klassendiagramme erstellen
- Sequenzdiagramme erstellen
- Zustandsdiagramme erstellen
- Möglichkeit Verwendung Design Patterns prüfen

Implementierung

- Klassen, Automaten, Operationen, Schnittstellen implementieren
- Traceability Matrix
- Code Metriken erstellen

Test

- Testfälle aufstellen, durchführen
- Code Review durchführen
- Testdoku

Nachbereitung

- Fehlerverfolgung
- Retro (evtl. neu aufgetauchte Requirements)

- LED (rot blinken: low battery)
  - überarbeiten: (rot leuchten: lädt)
  - überarbeiten: (grün leuchten: voll geladen)

Parameter in API Doku, Design & Architektur nachziehen

! Farbenblindheit in Sprint 2

Designentscheidung Farben

- Warnung: rot → kulturell etablierte Warnfarbe (aber schlecht erkennbar bei Farbsehschwäche)
- Ladeaktivität: gelb → gut unterscheidbar von rot & deutet Übergang zwischen Akku leer (rot) und Akku voll (typisch grün) an
- Ladeabschluss: blau → für nahezu alle Formen der Farbsehschwäche gut erkennbar

Anzeigedesign Ladeaktivität

- LED leuchtet währenddessen durchgehend gelb + auf Knopfdruck SoC anzeigen, aber Anschalten (Betrieb) nicht möglich

Anzeigedesign Ladeende

- LED blinkt langsam blau (solange wie Ladekabel noch dran, Anschalten (Betrieb) nicht möglich, aber Display schon)

Ladefehler & wie diese angezeigt werden

- Temperaturüberschreitung: kein Weiterladen (Keine besondere Anzeige, nur ●, einfach weiterladen wenn Temperatur ok
- Keine Eingangsspannung erkannt (z.B. wegen defektem Kabel): LED zeigt keine Ladeaktivität an

Eingangssignale für Ladeerkennung

- in Realität: Eingangssignal von Ladegerät (Adapter) über Kabel
- hier: Eingabe über Kommandozeile ("Start", "Stopp")

energieeffizientes Design

- LED & keine zusätzliche Anzeige

Option für multisensorische Warnsignale prüfen

- sinnvoll visuell & einmal (bei Grenze → Zustandsübergang) akustisch (man bekommt Warnung ug. niedrigem SoC auf jeden Fall mit → gerade wichtig für Barrierefreiheit

Überladeschutzlogik

- wenn man einen Akku zu lange lädt, altert er schneller und verliert so an Lebensdauer
- wenn 100% aufgeladen sind wird Verbindung zu Ladegerät getrennt (als wäre ausgestöpselt)

Temperatursimulator - Temperatursensor nötig

- Standardbetriebstemp.: 25 ± 2°C (als default Parameter, für Tests aber festlegen möglich)
- Standardladetemp.: 20-30°C (für Tests setzbar)

Regeln Ladeverhalten bei Temperaturänderungen

- < 15°C & > 45°C Ausschalten (Betrieb) oder Ladevorgang anhalten bis passende Temperatur

Schnittstellen

BatteryStateController(), calculatesSoC(), isLow() → von batterylogic in userInterface

InteractionHandler(), get/setDisplayState() → in UI ! handleButtonPress BL → HA

updateOperationState() → in UI

! TemperatureSim(), run() → in Programm ok weil Simulation

! ButtonInput() → in UI

Tasks Sprint 3:

LED (rot blinken: low battery)
überarbeiten: (rot leuchten: lädt)
(grün leuchten: voll geladen)

Recherche

Recherche aufgaben → Ergebnisse dokumentieren

Architektur

Komponentendiagramm erstellen

Schnittstellendefinition erstellen

Design

Klassendiagramme erstellen

Sequenzdiagramme erstellen

Zustandsdiagramme erstellen

Möglichkeit Verwendung Design Patterns prüfen

Implementierung

Klassen, Automaten, Operationen, Schnittstellen implementieren
Traceability Matrix | Designpatterns umsetzen
Code Metriken erstellen, Single-Responsibility umsetzen

Test

Testfälle aufstellen, durchführen

Code Review durchführen
Test Doku

Nachbereitung

Sprint Datei dokumentieren

Testfälle in Traceability Matrix

Retro

Benutzerschnittstelle (Bedienung, Anzeige)

Lessons Learned

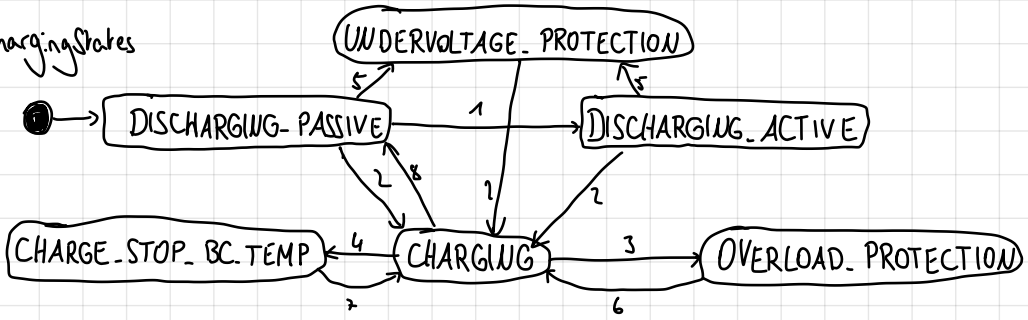
Readme zusammenbauen
Gesamtdoku-File, Erklärung

Hilfsmittel Verwendung dokumentieren

Abgabe

Schritte für Vorbereitung auf Präsi definieren

ChargingStates



1: Betrieb des Rasierers (OperationState = OPERATING

2: Ladevorgang gestartet

3: Ladung voll (100 %)

4: zu hohe Temperatur (≥ 45°)

5: kritische Unterspannung erreicht/Unterschritten (< 2,8V) → Dauerrotleuchten 4 in Stromsparmodus

6: Laden unterbrochen

7: Temperatur ok (< 45°)

8: Ladevorgang beendet (Kabel entfernt)