IHK Berlin

IT-BERUFE

Protokoll über die betriebliche Projektarbeit

Diese Erklärung ist der Dokumentation nach dem Deckblatt beizufügen.

Hiermit versichere ich,

Till Tomczak

Vor- und Nachname des Prüfungsteilnehmers

die betriebliche Projektarbeit

MYP – Manage Your Printer

Genaue Bezeichnung der Projektarbeit/ Prüfungsausschuss

sowie die eingereichte Dokumentation unter Betreuung von

Martin Noack / Mercedes-Benz AG

Vor- und Nachname des Ausbilders/ Name des Ausbildungsbetriebes

selbstständig und **ohne fremde Hilfe** konzipiert, verfasst und durchgeführt zu haben. Teile der Dokumentation, die ich **nicht selbstständig** erstellt habe, sind von mir entsprechend **gekennzeichnet** worden. Darüber hinaus erkläre ich, dass ich bei der vorliegenden Dokumentation **keine IT-/KI-gestützten Schreibwerkzeuge** genutzt habe. Die von der Verordnung vorgesehene Richtzeit wurde eingehalten. Die Arbeit hat in dieser Form oder ähnlicher Form keiner anderen Prüfungsinstitution vorgelegen.

Ich bin darüber aufgeklärt worden, dass meine betriebliche Projektarbeit bei **Täuschungshandlungen, bzw. Ordnungsverstößen mit „Null" Punkten bewertet** wird und als nicht bestanden gilt.

Ich bin weiter darüber aufgeklärt worden, dass dies auch dann gilt, wenn festgestellt wird, dass meine Projektdokumentation im Ganzen oder zu Teilen mit der eines anderen Prüfungsteilnehmers übereinstimmt. Mit der Durchführung einer Plagiatsprüfung und einer Prüfung, ob IT-/KI-gestützte Schreibwerkzeuge eingesetzt wurden, erkläre ich mich einverstanden.

Arbeitszeit

Das Projekt wurde von mir in der kalkulierten Zeit komplett fertiggestellt, einschließlich erforderlicher Nacharbeit ☐ ja ☒ nein

Nein, die Zeit wurde um 2 Stunden ☐ unterschritten ☒ überschritten.

Fehleranalysen und Einarbeitung in Programmiersprache bestehender Systeme

Begründung

Persönliche Erklärung

Ich versichere durch meine Unterschrift, dass ich die Projektarbeit und die eingereichte Dokumentation selbstständig angefertigt, alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen, als solche kenntlich gemacht habe.

04.06.2025

Datum

Till Tomczak

Unterschrift Prüfungsteilnehmer/-in

Unterschrift Projektverantwortliche/-r des Auftraggebers

Abschlussprüfung - Sommer 2025

Fachinformatik für digitale Vernetzung -
Dokumentation der betrieblichen Projektarbeit

Abgabedatum: 5. Juni 2025

MYP – Manage Your Printer

Vernetzte 3D-Druck-Reservierungsplattform mit IoT-Anbindung und zentraler Verwaltungsoberfläche

Prüfungsbewerber

Till Tomczak
Hainbuchenstraße 19
D-16761 Hennigsdorf

Ausbildungsbetrieb

Mercedes-Benz AG
Daimlerstraße 143
D-12277 Berlin

Mercedes-Benz

Inhaltsverzeichnis

© 2025 - Till Tomczak

Kurzglossar

Flask	Python-basiertes Web-Framework für Backend-Entwicklung. Grundgerüst des MYP-Servers mit REST-API und Session-Management
Raspberry Pi	Einplatinencomputer als zentrale Serverplattform. Upgrade von Pi 4 auf Pi 5 wegen Performance-Anforderungen
REST	Architekturstil der Kommunikationsschnittstelle (= API) für verteilte Systeme basierend auf HTTP-Protokoll
Scheduler	Zeitgesteuerte Subroutine für Smart-Plug-Operationen
Smart-Plug	Netzwerkfähige Steckdose zur Fernsteuerung elektrischer Geräte; nachfolgend: TP-Link TAPO P110 als Hardware-Schnittstelle
SQLite	Serverlose Ein-Dateien-Datenbank
TAPO	Tochterfirma des chinesischen Mutterkonzerns TP-Link, entwickelt Smart-Home Produkte
TBA	Technische Berufsausbildungsstätte (genau: Bau 26) der Mercedes-Benz AG Werk Berlin

© 2025- Till Tomczak

1. Einleitung

1.1 Analyse des Projektauftrages

Die Technische Berufsausbildungsstätte (TBA) der Mercedes-Benz AG am Standort Berlin verfügt über sechs 3D-Drucker verschiedener Hersteller (Prusa, Anycubic; B-Ware im Vergleich zu 3D-Druckern von Kostenstellen höherer Priorität sozusagen). Diese Geräte stellen eine wichtige Ressource für die praktische Ausbildung dar, weisen jedoch erhebliche technische Limitierungen auf; beispielsweise verfügen die Drucker weder über Funk- noch Netzwerkschnittstellen oder andere gesamteinheitliche Steuerungsmöglichkeiten. Diese technischen Einschränkungen verhinderten bislang eine koordinierte digitale Verwaltung und eine damit einhergehende Übersicht von Reservierungen und Nutzungsplänen der Azubis.

Das ‘bestehende Reservierungssystem' - wenn man es nun so nennen kann - basierte auf einem analogen Whiteboard, welches neben den Druckern positioniert war. Dies führte zu systematischen Problemen: Doppelbuchungen traten regelmäßig auf, wenn mehrere Nutzer zeitgleich Reservierungen vornahmen, die manuelle Aktivierung und Deaktivierung der Geräte wurde häufig versäumt - was zu unnötigem Energieverbrauch und erhöhtem Verschleiß führte - und eine verlässliche Dokumentation der tatsächlichen Nutzungszeiten existierte nicht, wodurch weder aussagekräftige Betätigungs- und Verantwortungszuordnung (bspw. für Aufräumarbeiten), noch eine verursachungsgerechte Kostenzuordnung möglich waren.

Ein erstmaliger Lösungsansatz durch den ehemaligen Auszubildenden Torben Haack hatte einen vielversprechenden Frontend-Prototyp auf Basis von Next.js hervorgebracht. Der Prototyp verfügte über eine moderne Benutzeroberfläche und gute Analysefunktionen, allerdings jedoch fehlte ganz fundamental die essentielle Backend-Funktionalität; ohne dies blieb die auf Prototypen-basierende Projektarbeit des Torben Haacks in der praktischen Anwendung ohne jegliche Funktion. Ich sah für mich also die Chance, die Idee hinter dem Prototypen aufzugreifen und mich ihrer im Rahmen meiner hier dargelegten Projektarbeit anzunehmen, da ich sofort mehrere Möglichkeiten zur Einbringung meiner Fachrichtung identifizieren konnte und ich keine notwendige Obligation - wie bei anderen Projektmöglichkeiten die sich mir boten - verspürte, sondern einen Anflug von Ideen, Tatendrang und intrinsischer Motivation; sprich: es kitzelte meine Leidenschaft.
LASTENHEFT / PFLICHTENHEFT

1.2 Ableitung der Projektziele

Nach erfolgter Zulassung des Abschlussprojekts durch die IHK kristallisierten sich die Projektziele in ihrer ganzen Komplexität heraus. Das zu entwickelnde System sollte unter dem prägnanten Namen "MYP - Manage Your Printer" nicht nur die digitale Verwaltung der Reservierungen ermöglichen, sondern – und hier liegt die besondere Herausforderung für einen Fachinformatiker der digitalen Vernetzung – auch die automatisierte Steuerung der physischen Geräte realisieren.

© 2025 - Till Tomczak

Die zentrale technische Herausforderung bestand in der Überbrückung der technischen Limitierungen der vorhandenen 3D-Drucker. Da eine direkte Kommunikation mit den Geräten aufgrund fehlender Schnittstellen nicht möglich war, musste ein alternativer, kreativer Ansatz zur Hardware-Steuerung entwickelt werden. Gleichzeitig waren die strengen unternehmensinternen Sicherheitsrichtlinien zu berücksichtigen, die keine direkten, geschweige denn permanenten Internetverbindungen in der Produktionsumgebung gestatten – eine Anforderung, die das Projekt zusätzlich verkomplizierte.

Ein weiteres, nicht zu unterschätzendes Projektziel war die Gewährleistung der Herstellerunabhängigkeit. Die heterogene und schnittstellenarme Druckerlandschaft der sechs 3D-Drucker erforderte eine universell einsetzbare Lösung, die sich zugleich auch leicht an zukünftige Upgrades – sowohl der 3D-Drucker als auch der entstehenden Lösung selbst – anpassen lassen würde. Das System sollte zudem eine rudimentäre, aber effektive Rechteverwaltung implementieren, die zwischen administrativen Funktionen und regulären Nutzerrechten differenziert.

1.3 Projektabgrenzung

Der Projektumfang wurde – durchaus pragmatisch, möchte man meinen – auf die praktische Umsetzung einer funktionsfähigen Lösung fokussiert. Eine umfassende Daten- und Prozessanalyse wurde bewusst zugunsten der technischen Realisierung zurückgestellt; diese Priorisierung ermöglichte die Fertigstellung eines produktiv einsetzbaren Systems innerhalb des knapp bemessenen Zeitrahmens von fünf Wochen.

Eine direkte Kommunikation mit den 3D-Druckern zur Übertragung von Druckdaten oder zur Statusüberwachung wurde kategorisch aus dem Projektumfang ausgeschlossen. Die fehlenden technischen Schnittstellen der vorhandenen Geräte hätten umfangreiche Hardware-Modifikationen erfordert, die weder zeitlich noch wirtschaftlich vertretbar gewesen wären – ganz zu schweigen von den Garantieverlusten, die solche Eingriffe unweigerlich nach sich gezogen hätten.

Die Integration in das unternehmensweite Intranet war ursprünglich fest eingeplant – ein Vorhaben, das sich als verhängnisvoller Trugschluss erweisen sollte. Zur Projektmitte hatte ich die bereits genehmigten SSL-Zertifikate des Haack'schen Prototyps durch einen unglücklichen Neuinstallationsprozess unwiederbringlich gelöscht; ein Moment des Schreckens, der die gesamte Projektplanung ins Wanken brachte. Immerhin war ich so weit gekommen, dass ich vom Frontend aus den GitHub OAuth-Zertifizierungsmechanismus ansteuern konnte – doch eine uns im E-Mail-Verkehr zuvor mitgeteilte IP-Adresse war aus irgendeinem Grund im DNS nicht mehr richtig zugeordnet, wie ich mit Zenmap (NMap-Gui) herausfand. Die Intranetanbindung blieb somit ausstehend; zum Zeitpunkt der Abgabe war sie aufgrund der Konzerngröße und der damit einhergehenden, entschleunigenden Formalitäten und Genehmigungsprozesse unvollkommen. Diese Anbindung hätte zusätzliche Sicherheitsprüfungen erfordert, die den bereits strapazierten Projektrahmen endgültig gesprengt hätten. Stattdessen wurde eine autarke Lösung entwickelt, die alle erforderlichen Funktionen lokal bereitstellt – ein Ansatz, der sich trotz der Rückschläge als gangbar erwies.

© 2025 - Till Tomczak

1.4 Projektumfeld

Das Projekt wurde im Rahmen meiner Ausbildung zum Fachinformatiker für digitale Vernetzung bei Mercedes durchgeführt. Die Technische Berufsausbildungsstätte bot dabei die vorhandene Infrastruktur und – wenn auch manchmal zögerliche – fachliche Unterstützung durch die Ausbildungsleitung.

Da Torben Haack seine Ausbildung bereits abgeschlossen hatte, als ich nach offizieller IHK-Zulassung mit der Projektarbeit begann, konnte ich auf seinen bereits existierenden Frontend-Prototyp aufbauen. Es handelte sich dabei um eine rein sequenzielle Weiterentwicklung ohne vorherige Abstimmung oder Zusammenarbeit – ich übernahm lediglich das vorhandene Artefakt und erweiterte es zu einer Gesamtlösung. Diese Konstellation erwies sich als Segen und Fluch zugleich.

Die organisatorischen Rahmenbedingungen wurden maßgeblich durch die konzerninternen Sicherheitsrichtlinien und IT-Doktrinen geprägt. Jede technische Entscheidung musste die Vorgaben bezüglich Netzwerksicherheit, Datenschutz und Compliance berücksichtigen. Die Beantragung notwendiger Administratorrechte und die Genehmigung selbstsignierter SSL-Zertifikate erforderten umfangreiche Abstimmungsprozesse mit der IT-Abteilung – Prozesse, die sich teilweise über Wochen hinzogen und meine Geduld auf eine harte Probe stellten.

1.5 Betriebliche Schnittstellen

Die Analyse der betrieblichen Schnittstellen offenbarte ein komplexes Geflecht von Abhängigkeiten und Anforderungen. Primär musste das System mit der bestehenden Netzwerkinfrastruktur der TBA harmonieren, ohne dabei Sicherheitsrichtlinien zu verletzen. Die Schnittstelle zur IT-Abteilung erwies sich als besonders kritisch, da jede Netzwerkkonfiguration und jede Port-Freischaltung einer expliziten Genehmigung bedurfte.

Die Benutzerschnittstelle musste so gestaltet werden, dass sowohl technisch versierte Auszubildende als auch weniger IT-affine Nutzer das System intuitiv bedienen können. Dies erforderte eine Balance zwischen Funktionsumfang und Benutzerfreundlichkeit.

Besonders herausfordernd gestaltete sich die Schnittstelle zu den Smart-Plugs. Die ursprüngliche Annahme, dass sich die TAPO P110-Steckdosen von TP-Link problemlos integrieren lassen würden, erwies sich als zu optimistisch. Die Geräte boten keine dokumentierte API – nur die proprietäre TAPO-App ermöglichte die Steuerung. Dies stellte eine erhebliche technische Herausforderung für die geplante Integration dar.

1.6 Analyse der IT-sicherheitsrelevante Bedingungen

Die Sicherheitsanalyse offenbarte multiple Herausforderungen, die es zu bewältigen galt. Das System musste in einem isolierten Netzwerksegment betrieben werden, ohne dabei die Funktionalität einzuschränken. Die Anforderung, keine permanente Internetverbindung zu etablieren, schloss Cloud-basierte Lösungen kategorisch aus – ein Umstand, der die Auswahl geeigneter Smart-Plugs erheblich einschränkte und mich zu kreativen Lösungsansätzen zwang.

© 2025 - Till Tomczak

Die Authentifizierung und Autorisierung musste robust implementiert werden, ohne die Benutzerfreundlichkeit zu beeinträchtigen – ein klassisches Dilemma der IT-Sicherheit. Die Entscheidung für bcrypt-basiertes Password-Hashing stellte einen vernünftigen Kompromiss zwischen Sicherheit und Performance auf dem ressourcenbeschränkten Raspberry Pi dar; die Details der Implementierung überließ ich – naturgemäß außerhalb meiner Kernkompetenz der digitalen Vernetzung liegend – der bewährten Flask-Login-Bibliothek.

Besondere Aufmerksamkeit erforderte die Absicherung der API-Endpunkte. Jeder Endpunkt musste gegen gängige Angriffsvektoren wie SQL-Injection, Cross-Site-Scripting und CSRF-Attacken geschützt werden. Die Implementierung eines Rate-Limiting-Mechanismus erschwert Brute-Force-Angriffe auf die Authentifizierungsschnittstelle – eine Maßnahme, die zwar keinen absoluten Schutz bietet, aber die Hürde für Angreifer signifikant erhöht.

1.7 Darstellung der vorhandenen Systemarchitektur

Wenn man ein wenig schielt und das vorgefundene Haack'sche Projekt als Systemarchitektur betrachtet – sah man sich diffusen Komponenten entgegen, deren Kernkomponente ein Raspberry mit Netzteil an einem vollkommen willkürlich platzierten Ort irgendwo in der TBA war. Die 3D-Drucker operierten als Insellösungen, verbunden lediglich durch ihre physische Nähe und das gemeinsame Whiteboard. Der Frontend-Prototyp von Torben existierte als Docker-Container auf jenem Raspberry, operativ abgekapselt ohne jegliche praktische Integration – ohne Anbindung an reale Daten oder sonstiges; als Darstellungsprojekt erreichbar jedoch vom Intranet aus, das muss festgehalten sein und wird sich im Verlaufe meines Projektes ändern – wie sich herausstellte.

Die Netzwerkinfrastruktur der TBA basierte auf einem segmentierten Ansatz mit verschiedenen VLANs für unterschiedliche Geräteklassen. Die 3D-Drucker waren – mangels Netzwerkfähigkeit – nicht in diese Struktur integriert. Der bereitgestellte Raspberry Pi 4 (der sich später als unterdimensioniert erweisen und durch einen Pi 5 ersetzt werden sollte) fungierte als zentrale Plattform für das MYP-System.

Die Analyse ergab, dass eine grundlegende Neukonzeption der Architektur erforderlich war. Die Lösung musste die isolierten Komponenten zu einem kohärenten System verbinden, ohne dabei die bestehenden Sicherheitsrichtlinien zu verletzen. Der gewählte Ansatz – die Steuerung über Smart-Plugs – stellte einen eleganten Kompromiss zwischen technischer Machbarkeit und praktischem Nutzen dar; eine Entscheidung, die nicht zuletzt auf meiner privaten Erfahrung mit TAPO-Geräten basierte. In meiner privat geführten Infrastruktur hatte ich bereits TAPO-Geräte aller Art integriert – stets ging dies recht schnell und einfach vonstatten. Privat nutzte ich ebenfalls Air-Gapped-Networks hierfür, jedoch mit dem entscheidenden Unterschied, nicht mit der eigenständigen programmatischen Integration eben jener Geräte als Hauptkomponente beauftragt zu sein; ein Unterschied, der sich als gravierend herausstellen sollte.

© 2025 - Till Tomczak

2. Projektplanung

2.1 Terminplanung

Die Projektplanung folgte einem agilen Ansatz nach Scrum-Prinzipien – eine Entscheidung, die sich angesichts der zahlreichen Unwägbarkeiten als richtig erweisen sollte. Die Gesamtprojektdauer von fünf Wochen (15. April bis 20. Mai 2025) wurde in fünf einwöchige Sprints unterteilt, wobei jeder Sprint seine eigenen Herausforderungen und – ja – auch Überraschungen bereithielt.

Sprint 1 (15.-19. April 2025)

Der erste Sprint widmete sich der Analyse des vorgefundenen Prototyps und der Definition der Erweiterungspunkte. Nach Projektstart und erstmaliger Sichtung der Frontend-Codebasis offenbarte sich eine solide, wenn auch stellenweise überkomplexe Struktur. Die Spezifikation der erforderlichen API-Endpunkte gestaltete sich umfangreicher als erwartet – über 100 Endpunkte wurden identifiziert, was mich zunächst erschaudern ließ. Der kritische Meilenstein dieses Sprints war die erfolgreiche Etablierung der Kommunikation mit den Smart-Plugs über die PyP100-Bibliothek; ein Vorhaben, das zunächst kläglich scheiterte.

Sprint 2 (22.-26. April 2025)

Im zweiten Sprint lag der Fokus auf dem Aufbau der Backend-Infrastruktur. Die Beantragung der erforderlichen Administratorrechte erwies sich als zeitaufwändiger als erwartet. Parallel dazu begannen die ersten Experimente mit Wireshark, um das Kommunikationsprotokoll der Smart-Plugs zu entschlüsseln – eine Notwendigkeit, die sich aus der mangelhaften Dokumentation der PyP100-Bibliothek ergab.

Sprint 3 (29. April- 3. Mai 2025)

Der dritte Sprint sollte die Integration von Frontend und Backend realisieren. Stattdessen mutierte er zu einer Woche der technischen Herausforderungen: Die Verbindung zwischen den Komponenten scheiterte wiederholt, die genehmigten SSL-Zertifikate gingen durch einen unglücklichen Neuinstallationsprozess verloren und somit auch der hart erarbeitete Intranet-Zugang, bei dem ich Torben damals – unwissend meines späteren Projektes – noch zur Seite stand. Zudem kostete die Einarbeitung in die unternehmensspezifischen Implementierungen von GitHub OAuth und npm weiterhin wertvolle Zeit.

Sprint 4 (6.-10. Mai 2025)

Ursprünglich für Optimierungen vorgesehen, passierte in dem Sprint alles: von Krisensitzung bzgl. Des Umgangs mit der vorliegenden Situation bis hin zur Ausarbeitung einer Rettungsmission und der Umsetzung dieser. Der Zeitdruck erzwang pragmatische Entscheidungen und die Konzentration auf essenzielle Funktionen. In intensiven Coding-Sessions wurde die Grundfunktionalität – sprich die Ansteuerung der Steckdosen per Python-Code mit Rudimentärem Flask-Frontend – implementiert; nicht unbedingt elegant, aber funktional.

© 2025- Till Tomczak

Sprint 5 (13.-17. Mai 2025)

Der finale Sprint diente der Fehlerbehebung und Systemstabilisierung. Die ursprünglich geplanten ausführlichen Schulungen fielen mehr oder minder dem Zeitdruck zum Opfer, wurden aber dennoch in gut improvisierter Form durchgeführt, sodass die Kollegen / Ausbilder ein intuitives Verständnis entwickeln konnten. Weiterhin wurde an kritischen Bugfixes gearbeitet und die Projektdokumentation erstellt.

2.2 Ressourcenplanung

Die Ressourcenplanung gestaltete sich als Balanceakt zwischen technischen Anforderungen und budgetären Beschränkungen. Die Hardware-Ausstattung wurde sorgfältig – und doch pragmatisch – ausgewählt.

Als zentrale Serverplattform diente zunächst ein Raspberry Pi 4 mit 4 GB RAM – eine Entscheidung, die sich schnell als Fehlkalkulation herausstellte. Performance-Tests zeigten, dass das ressourcenhungrige Next.js-Frontend die kleine Platine an ihre Grenzen brachte. Der kurzfristige Wechsel auf einen Raspberry Pi 5 mit 8 GB RAM und 128 GB Speicher löste die Performance-Probleme, verursachte aber zusätzliche Kosten und – was schwerer wog – Zeitverzug.

Die sechs TP-Link Tapo P110 Smart-Plugs bildeten das Herzstück der Hardware-Lösung. Jedes Gerät erhielt eine statische IP-Adresse im Bereich 192.168.0.100 bis 192.168.0.106 – eine scheinbar triviale Konfiguration, die sich später als entscheidend für die Systemstabilität erweisen sollte. Die ursprüngliche Annahme, dass sich diese Geräte problemlos integrieren lassen würden, erwies sich als optimistisch; die proprietäre API erforderte erheblichen Reverse-Engineering-Aufwand mittels Wireshark.

PREISKALKULATION

Zur professionellen Unterbringung der Hardware wurde ein 19-Zoll-Serverschrank beschafft. Die internen Beschaffungsprozesse erwiesen sich jedoch als so langwierig, dass ergänzende Komponenten wie Lüftereinheiten und Kabelmanagement-Systeme aus eigener Tasche finanziert wurden – eine Investition in die professionelle Präsentation des Projekts, die mir wichtig war.

Die Software-Architektur basierte vollständig auf Open-Source-Technologien: Python 3.11 als Programmiersprache, Flask 2.3 als Web-Framework, SQLAlchemy 2.0 für die Datenbankabstraktion und SQLite als Datenbanksystem. Diese Technologieauswahl gewährleistete nicht nur Unabhängigkeit von proprietären Lösungen, sondern erfüllte auch die strikte Offline-Anforderung des Projekts – ein Umstand, der sich als Segen erwies.

Als Betriebssystem stand zunächst eine Entscheidung zwischen OpenSUSE und NixOS im Raum – NixOS schien durch seine deklarative Konfiguration für diesen Einsatzzweck prädestiniert. Letztendlich entschied ich mich doch für Raspbian, um nicht unnötig zu experimentieren und die knapp bemessene Zeit effizient zu nutzen; Pragmatismus über technische Eleganz.

© 2025 - Till Tomczak

2.3 Planung der Qualitätssicherung

Das Qualitätssicherungskonzept orientierte sich am V-Modell. Für jede Entwicklungsphase wurden korrespondierende Testaktivitäten definiert.

Zur Gewährleistung realistischer Testbedingungen wurde eine dedizierte Testumgebung mittels VirtualBox etabliert. Ursprünglich war die Implementierung zweier virtueller Maschinen vorgesehen – eine für das Backend, eine für das Frontend – um die geplante verteilte Architektur vollständig zu simulieren. Die zeitlichen Restriktionen erzwangen jedoch eine Fokussierung auf die Backend-Testumgebung. Diese virtuelle Maschine, basierend auf Debian mit Hardware-Konfigurationen analog zum Produktivsystem des Raspberry Pi, ermöglichte realitätsnahe Tests ohne Gefährdung der Produktivumgebung sowie die Gewährleistung meiner absolut-mobilen Produktivität.

Die Konfiguration der Testumgebung erforderte spezielle Anpassungen an die Unternehmensrichtlinien: Da Port 443 auf Dienstrechnern von Mercedes standardmäßig blockiert ist, wurde eine Port-Weiterleitung implementiert, die den Zugriff vom Host-System über alternative Ports ermöglichte. Diese Lösung gewährleistete vollständige Funktionstests bei gleichzeitiger Compliance mit den Sicherheitsrichtlinien.

Auf Unit-Test-Ebene wurden alle kritischen Komponenten isoliert getestet. Die Datenbankoperationen, API-Eingabevalidierung und Kernfunktionen der Reservierungsverwaltung durchliefen umfangreiche Testszenarien. Besondere Aufmerksamkeit galt der Smart-Plug-Kommunikation, für die spezielle Testfälle entwickelt wurden – einschließlich simulierter Netzwerkausfälle und Timeout-Situationen.

Die Integrationstests gestalteten sich komplexer als erwartet. Das Zusammenspiel zwischen Backend und Smart-Plugs erwies sich als fehleranfällig, insbesondere bei gleichzeitigen Zugriffen. Die systematischen Tests mit verschiedenen Eingabedaten – einschließlich bewusst invalider und potenziell schädlicher Inputs – deckten mehrere Sicherheitslücken auf, die nachträglich geschlossen werden mussten.

Systemtests bildeten komplette Anwendungsszenarien ab. Ein typischer Testfall umfasste die Benutzeranmeldung, Reservierungserstellung, automatische Druckeraktivierung zur geplanten Zeit und die anschließende Deaktivierung. Die zeitliche Präzision der Schaltungen – kritisch für die Benutzerzufriedenheit – wurde dabei besonders überwacht.

Performance-Tests auf der Zielplattform offenbarten die bereits erwähnten Limitierungen des Raspberry Pi 4. Der Wechsel auf den Pi 5 löste diese Probleme, erforderte aber eine Wiederholung aller Tests. Die finale Konfiguration bewältigte selbst simultane Zugriffe mehrerer Nutzer und parallele Smart-Plug-Operationen ohne nennenswerte Einbußen – ein Triumph der Optimierung.

2.4 Bewertung der heterogenen IT-Landschaft

Die IT-Landschaft der TBA präsentierte sich als bunter Flickenteppich verschiedenster Technologien und Standards. Die 3D-Drucker stammten von unterschiedlichen Herstellern

© 2025 - Till Tomczak

mit inkompatiblen Steuerungssystemen. Das Netzwerk war in multiple VLANs segmentiert, wobei die Dokumentation dieser Struktur bestenfalls als lückenhaft bezeichnet werden konnte.

Die Herausforderung bestand darin, eine einheitliche Lösung für diese heterogene Umgebung zu entwickeln. Der Ansatz über Smart-Plugs erwies sich hier als Glücksgriff – er abstrahierte die Unterschiede zwischen den Druckern auf die simpelste mögliche Ebene: Strom an oder aus. Diese radikale Vereinfachung ermöglichte eine universelle Lösung, die unabhängig vom Druckermodell funktionierte.

Die Integration in die bestehende Netzwerkinfrastruktur erforderte diplomatisches Geschick. Die IT-Abteilung bestand auf strikter Segmentierung, was die Kommunikation zwischen Komponenten verkomplizierte. Die Lösung – ein dediziertes IoT-Subnetz für das MYP-System – stellte einen akzeptablen Kompromiss dar, der sowohl Sicherheitsbedenken als auch funktionale Anforderungen berücksichtigte.

2.5 Anforderungsgerechte Auswahl der Übertragungssysteme

Die Auswahl der Übertragungssysteme wurde maßgeblich durch die Sicherheitsanforderungen bestimmt. Cloud-basierte Lösungen schieden kategorisch aus, was die Optionen erheblich einschränkte. Die Entscheidung für lokale HTTP/HTTPS-Kommunikation mit selbstsignierten Zertifikaten war pragmatisch, aber effektiv.

Die Kommunikation mit den Smart-Plugs stellte die zentrale technische Herausforderung dar. Die TAPO-Geräte boten keine dokumentierte Programmierschnittstelle – die Steuerung erfolgte ausschließlich über die proprietäre Hersteller-App. Eine systematische Protokollanalyse mittels Wireshark wurde daher unumgänglich. Die Untersuchung des Netzwerkverkehrs zwischen App und Steckdosen offenbarte eine verschlüsselte Kommunikation mit dynamisch generierten Session-Keys.

Mein initialer Implementierungsversuch mit einem recherchierten Python-Modul verlief erfolglos – die Kompatibilität mit den vorhandenen Geräten war nicht gegeben. Die Wireshark-Analyse zeigte konsistente verschlüsselte Response-Muster. Nach mehreren erfolglosen Versuchen, einzelne Anfragen zu replizieren, wurde deutlich: Die korrekte Sequenzierung der Kommunikation war essentiell. Das Protokoll nutzte temporäre Authentifizierungs-Cookies in Kombination mit proprietärer Verschlüsselung.

Nach intensiver Recherche und mehreren Tagen systematischer Tests konnte PyP100 als geeignete Lösung identifiziert werden. Dieses auf GitHub verfügbare Python-Modul implementierte das proprietäre Protokoll korrekt und ermöglichte eine stabile Integration der Smart-Plugs in die Systemarchitektur.

2.6 Planung der Prozess-/ und Systemschnittstellen

Die Schnittstellenplanung erforderte eine sorgfältige Balance zwischen Funktionalität und Sicherheit. Die REST-API wurde nach modernen Standards entworfen, mit klarer Trennung zwischen öffentlichen und authentifizierten Endpunkten. Über 100 Endpunkte wurden

© 2025 - Till Tomczak

spezifiziert – eine Anzahl, die zunächst umfangreich erschien, sich jedoch als notwendig für die vollständige Funktionsabdeckung erwies. Jeder Endpunkt wurde präzise auf seine spezifische Aufgabe zugeschnitten.

Die Schnittstelle zwischen Frontend und Backend basierte auf JSON-formatierter Kommunikation über HTTPS. Die Implementierung von CORS-Policies gestaltete sich komplexer als erwartet, da die Sicherheitsrichtlinien strikte Einschränkungen vorgaben. Die Lösung – eine Whitelist-basierte CORS-Konfiguration – erfüllte die Sicherheitsanforderungen ohne die Funktionalität einzuschränken. Diese Implementation stellte einen ausgewogenen Kompromiss zwischen Sicherheit und Anwenderfreundlichkeit dar.

Besondere Aufmerksamkeit erforderte die Scheduler-Schnittstelle. Der als eigenständiger Thread implementierte Scheduler musste nahtlos mit der Hauptanwendung kommunizieren, ohne dabei Race Conditions oder Deadlocks zu verursachen. Die Verwendung von Thread-sicheren Queues und explizitem Locking löste diese Herausforderung mit einer technisch eleganten Architektur.

2.7 Planung der IT-Sicherheitsmaßnahmen

Die Sicherheitsplanung folgte dem Prinzip "Security by Design" – ein Ansatz, der sich angesichts der sensiblen Umgebung als unerlässlich erwies. Jede Komponente wurde von Anfang an mit Sicherheit im Hinterkopf entwickelt.

Die Authentifizierung basierte auf bcrypt mit einem Cost-Faktor von 12 – ein Kompromiss zwischen Sicherheit und Performance auf dem Raspberry Pi. Session-Management wurde über Flask-Login realisiert, mit konfigurierbaren Timeout-Werten und sicheren Session-Cookies. Die Implementierung einer Brute-Force-Protection mit exponentieller Backoff-Strategie verhinderte automatisierte Angriffe.

Auf Netzwerkebene wurden restriktive Firewall-Regeln implementiert. Nur essenzielle Ports wurden geöffnet, ausgehende Verbindungen auf die IP-Adressen der Smart-Plugs beschränkt. Die Verwendung von iptables ermöglichte granulare Kontrolle über den Netzwerkverkehr.

Die API-Sicherheit umfasste Input-Validation, Output-Encoding und CSRF-Protection. Jeder Endpunkt wurde gegen die OWASP Top 10 abgesichert. Ein selbstentwickeltes Intrusion Detection System überwachte verdächtige Aktivitäten und sperrte bei Bedarf IP-Adressen temporär.

3. Durchführung und Auftragsbearbeitung

3.1 Prozess-Schritte und Vorgehensweise

Die Durchführung des Projekts glich einer technischen Expedition mit unerwarteten Wendungen und kreativen Lösungsansätzen. Die ursprünglich geplante lineare Vorgehensweise wich schnell einer iterativen, problemgetriebenen Herangehensweise.

© 2025 - Till Tomczak

3.1.1 Datenabfrage der Sensoren

Die "Sensoren" in diesem Kontext waren die Smart-Plugs – eine euphemistische Bezeichnung für Geräte, die sich als technisch anspruchsvoll in der Integration erwiesen. Meine initiale Recherche nach einem geeigneten Python-Modul zur Steuerung verlief erfolglos. Das identifizierte Modul erwies sich als inkompatibel mit den vorhandenen Geräten.

Daraufhin erfolgte eine Protokollanalyse mittels Wireshark. Die Aufzeichnung des Netzwerkverkehrs zwischen TAPO-App und Smart-Plugs offenbarte ein komplexes Authentifizierungsprotokoll: Die Kommunikation erfolgte verschlüsselt unter Verwendung von Session-Tokens mit dynamischer Generierung bei jeder Authentifizierung. Die implementierte Verschlüsselung basierte auf einer RSA-AES-Hybridarchitektur – eine bemerkenswerte Sicherheitsimplementierung für Geräte dieser Preisklasse, die jedoch die Integration erheblich verkomplizierte.

Nach mehrtägiger Analyse und verschiedenen Implementierungsversuchen identifizierte ich PyP100 – ein Python-Modul, das die erforderliche lokale Kommunikation mit den TAPO-Geräten beherrschte. Diese auf GitHub verfügbare Bibliothek löste die Session-Key-Problematik durch eine elegante Implementierung des proprietären Protokolls.

Die Implementierung der Datenabfrage erfolgte über eine selbstentwickelte Wrapper-Klasse, die die Komplexität der Kommunikation kapselte. Retry-Mechanismen mit exponentieller Backoff-Strategie sorgten für Robustheit bei Netzwerkproblemen. Die Abfrage umfasste nicht nur den Schaltzustand, sondern auch Metadaten wie Energieverbrauch und Signalstärke – Informationen, die sich später als wertvoll für das Monitoring erwiesen.

3.1.2 Verarbeiten der Daten

Die Datenverarbeitung folgte einem ereignisgesteuerten Ansatz. Der Scheduler-Thread prüfte im Minutentakt die Datenbank auf anstehende Aktionen und triggerte entsprechende Smart-Plug-Operationen. Die Herausforderung bestand darin, die Asynchronität der Hardware-Operationen mit der Synchronität der Datenbankzugriffe zu vereinen.

Die Lösung war ein Queue-basiertes System, das Kommandos pufferte und sequenziell abarbeitete. Dies verhinderte Race Conditions bei simultanen Zugriffen und gewährleistete die Konsistenz der Systemzustände. Jede Operation wurde in einer Transaktion gekapselt, mit Rollback-Mechanismen bei Fehlern.

Die Verarbeitung der Energiedaten ermöglichte interessante Einblicke in die Nutzungsmuster. Anomalien – wie ungewöhnlich hoher Stromverbrauch – konnten erkannt und gemeldet werden. Diese Funktion, ursprünglich nicht in der Projektspezifikation vorgesehen, entwickelte sich zu einem wertvollen Feature für die präventive Wartung. Die ungeplante Zusatzfunktionalität erweiterte den Nutzen des Systems signifikant über die reine Reservierungsverwaltung hinaus.

© 2025 - Till Tomczak

3.2 Abweichung, Anpassung und Entscheidungen

Die Projektdurchführung war geprägt von kontinuierlichen Anpassungen an die Realität. Die größte Abweichung vom ursprünglichen Plan war der Wechsel der Systemarchitektur von einer verteilten zu einer konsolidierten Lösung.

Ursprünglich war geplant, dass ich nur die API entwickle und diese mit dem existierenden Frontend auf einem separaten Raspberry Pi verknüpfe. Diese Architektur erwies sich als zu komplex – die unterschiedlichen Technologie-Stacks (Next.js vs. Python/Flask) und die Netzwerksegmentierung machten die Integration schwierig. Die Entscheidung, beide Komponenten auf einem einzigen Raspberry Pi zu konsolidieren, vereinfachte nicht nur die Architektur, sondern reduzierte auch Kosten und Stromverbrauch.

Der versehentliche Verlust der SSL-Zertifikate während einer Neuinstallation führte zur Implementierung eines robusten Backup-Systems. Kritische Konfigurationsdateien werden nun dreifach gesichert – eine Lektion, die schmerzhaft gelernt wurde. Der Verlust der bereits genehmigten Zertifikate des Haack'schen Prototyps zur Projektmitte war ein Moment des Schreckens; die mühsam erkämpften Genehmigungen, die etablierten Vertrauensstellungen – alles dahin durch einen unbedachten Moment während der Systemkonfiguration.

Die Entscheidung, von meinem ersten Python-Modul-Versuch zu PyP100 zu wechseln, fiel nach tagelangen frustrierenden Debugging-Sessions. Der Stolz, es mit dem ersten Modul schaffen zu wollen, wich dem Pragmatismus, eine funktionierende Lösung zu liefern. PyP100 – ironischerweise simpler und stabiler – rettete das Projekt.

3.3 Maßnahmen zur Qualitätskontrolle

Die Qualitätskontrolle erfolgte kontinuierlich und vielschichtig. Automatisierte Tests liefen bei jedem Commit, manuelle Tests ergänzten diese bei kritischen Funktionen. Die Herausforderung bestand darin, die Hardware-abhängigen Komponenten testbar zu machen.

Mock-Objekte simulierten die Smart-Plugs für Unit-Tests. Diese Mocks replizierten das Verhalten der echten Hardware, einschließlich typischer Fehlerszenarien wie Timeouts oder Verbindungsabbrüche. Die Test-Coverage erreichte 85% – die fehlenden 15% waren hauptsächlich UI-Code und Error-Handler, deren Test-Aufwand in keinem vernünftigen Verhältnis zum Nutzen stand.

Die VirtualBox-basierte Testumgebung ermöglichte umfassende Systemtests unter produktionsnahen Bedingungen. Die virtuelle Maschine replizierte die Konfiguration des Produktivsystems, wodurch potenzielle Inkompatibilitäten frühzeitig identifiziert werden konnten. Die implementierte Port-Weiterleitung umging die Restriktionen des Unternehmensnetzes und ermöglichte vollständige End-to-End-Tests inklusive HTTPS-Kommunikation.

Integrationstests mit echter Hardware deckten Probleme auf, die in der Simulation nicht auftraten. Timing-Issues bei simultanen Zugriffen, Memory-Leaks bei lang laufenden

© 2025 - Till Tomczak

Operationen, Race Conditions im Scheduler – all diese Probleme wurden iterativ identifiziert und behoben.

Die Implementierung eines Logging-Systems erwies sich als unschätzbar wertvoll. Jede Operation, jeder Fehler, jede Anomalie wurde protokolliert. Die Log-Analyse wurde zum wichtigsten Debugging-Tool, insbesondere bei sporadisch auftretenden Problemen.

3.4 Implementierung, Konfiguration und Inbetriebnahme von Schnittstellen und unterschiedlicher Prozesse und Systeme

Die Implementierung der verschiedenen Schnittstellen erfolgte modular und iterativ. Die REST-API wurde Blueprint-basiert strukturiert, was eine klare Trennung der Funktionsbereiche ermöglichte. Authentication, User Management, Printer Management und Job Management erhielten jeweils eigene Blueprints.

Die Smart-Plug-Schnittstelle durchlief mehrere Iterationen. Die finale Implementation kapselte die gesamte Kommunikationslogik in einer einzigen Klasse, die eine simple API bot: turn_on(), turn_off(), get_status(). Diese Abstraktion verbarg die Komplexität des darunterliegenden Protokolls und ermöglichte einfache Erweiterungen. Die Datenbank-Schnittstelle nutzte SQLAlchemy's ORM-Funktionalität. Die Definition der Models erfolgte deklarativ, Migrationen wurden über Alembic verwaltet. Die Entscheidung für SQLite als Datenbank war pragmatisch – keine zusätzlichen Services, keine Konfiguration, perfekt für die Offline-Anforderung.

Der Scheduler wurde als eigenständiger Thread implementiert, der beim Anwendungsstart initialisiert wurde. Die Kommunikation mit dem Hauptthread erfolgte über thread-sichere Queues. Diese Architektur ermöglicht asynchrone Hardware-Operationen ohne Blockierung der Web-Requests.

3.5 Konfiguration von Übertragungssystemen und Integration in die Gesamtinfrastruktur

Die Integration in die Unternehmensinfrastruktur erforderte zahlreiche Kompromisse und kreative Lösungen. Das dedizierte IoT-Subnetz wurde speziell für das MYP-System eingerichtet, mit restriktiven Firewall-Regeln und ohne Internet-Zugang.

Die Netzwerkkonfiguration erfolgte in enger Abstimmung mit der IT-Abteilung. Jede Änderung erforderte ein Change-Request, jede Port-Öffnung eine Security-Review. Der bürokratische Overhead war erheblich, aber notwendig für die Compliance.

Die SSL-Konfiguration mit selbstsignierten Zertifikaten war ein notwendiges Übel. Ohne Internet-Zugang war Let's Encrypt keine Option. Die Zertifikate wurden mit OpenSSL generiert und mit allen relevanten SANs (Subject Alternative Names) versehen, um Kompatibilitätsprobleme zu vermeiden. Die Browser-Warnungen wurden durch eine dokumentierte Prozedur zur Zertifikats-Installation umgangen – nicht elegant, aber funktional.

© 2025 - Till Tomczak

Die Integration der Smart-Plugs erforderte statische IP-Adressen – DHCP-Reservierungen waren in der Netzwerk-Policy nicht vorgesehen. Die manuelle Konfiguration jedes Geräts war zeitaufwendig, gewährleistete jedoch stabile und vorhersagbare Netzwerkverbindungen.

Für die Administration und Wartung des Systems wurden Remote-Zugriffsmöglichkeiten implementiert. Das Setup-Skript konfigurierte automatisch SSH und RDP-Dienste, wodurch eine sichere Fernwartung des Raspberry Pi ermöglicht wurde. Diese Remote-Zugänge erwiesen sich als essentiell für die effiziente Systemadministration, insbesondere da der physische Zugang zum Serverschrank oft eingeschränkt war.

3.6 Erfüllen der Anforderungen an die Informationssicherheit

Die Informationssicherheit wurde von Anfang an als kritischer Erfolgsfaktor behandelt. Jede Designentscheidung wurde durch die Sicherheitsbrille betrachtet, jede Implementierung auf Schwachstellen geprüft.

Die Authentifizierung implementierte moderne Best Practices: bcrypt-Hashing, sichere Session-Verwaltung, CSRF-Protection. Die API-Endpunkte wurden systematisch gegen die OWASP Top 10 abgesichert. Input-Validation erfolgte auf mehreren Ebenen – Client-seitig für UX, Server-seitig für Sicherheit.

Die Implementierung eines Rate-Limiters verhinderte Brute-Force-Angriffe. Nach fünf fehlgeschlagenen Login-Versuchen wurde die IP-Adresse für 30 Minuten gesperrt – lang genug, um Angriffe unwirtschaftlich zu machen, kurz genug, um legitime Nutzer nicht übermäßig zu frustrieren.

DSGVO-Compliance wurde durch Privacy-by-Design erreicht. Personenbezogene Daten wurden minimiert, Löschfristen implementiert, Datenexport-Funktionen bereitgestellt. Die Logging-Funktionalität anonymisierte IP-Adressen nach 30 Tagen automatisch.

4. Projektabschluss

4.1 Soll-Ist-Vergleich (Abweichung, Anpassungen)

Der Vergleich zwischen geplanten und erreichten Zielen offenbart ein gemischtes, aber letztendlich positives Bild. Die Kernfunktionalität – digitale Reservierungsverwaltung mit automatischer Hardware-Steuerung – wurde vollständig implementiert und übertraf in einigen Aspekten sogar die ursprünglichen Anforderungen.

Erfolgreich umgesetzte Anforderungen

Das Produkt dieser Projektarbeit bietet eine vollständig digitalisierte Reservierungslösung mit automatischer Druckersteuerung via Smart-Plugs, robuster Authentifizierung, einer umfangreichen REST-API, offline-fähiger Architektur ohne Cloud-Abhängigkeiten sowie DSGVO-konformer Datenhaltung und integriertem Energiemonitoring mit Nutzungsstatistiken. Es wurde erfolgreich

Abweichungen vom ursprünglichen Plan:

© 2025 - Till Tomczak

Konsolidierung auf einen statt zwei Raspberry Pis

Wechsel von PyP100 zu alternativem Kommunikationsmodul

Hardware-Upgrade vom Pi 4 auf Pi 5

Verschiebung der Benutzerschulungen auf Nach-Projektphase

Die größte positive Überraschung war die erfolgreiche Integration des Energiemonitorings. Diese ursprünglich nicht geplante Funktion ermöglicht detaillierte Einblicke in Nutzungsmuster und Energieverbrauch – wertvolle Daten für die Optimierung des Druckerbetriebs.

Für die programmatische Umsetzung des Frontends nahm ich gänzlich Unterstützung künstlicher Intelligenz zu Hilfe – mehr als absolut notwendig, um das Zeitlimit nicht um Längen zu überschreiten und die Profession meiner Fachrichtung einzuhalten. Die Frontend-Entwicklung lag außerhalb meines Kernkompetenzbereichs der digitalen Vernetzung; die KI-Assistenz ermöglichte es mir, mich auf die Backend-Integration und Hardware-Anbindung zu konzentrieren – meine eigentlichen Stärken.

Die Implementierung eines Kiosk-Modus für die Werkstatt-Terminals erforderte zusätzliche Systemkonfiguration: Openbox als minimalistisches Desktop-Environment, Chromium im Kiosk-Modus mit automatischem Start dreier Instanzen – eine auf Port 443, eine auf Port 80 als Fallback für die API, sowie eine lokale Instanz auf Port 5000 für den Kiosk-Modus. Die Unternehmens-Root-CA-Zertifikate mussten manuell installiert werden; ein Shell-Skript automatisierte diesen Prozess, eine systemd-Service-Datei gewährleistete den Autostart. FirewallD diente als Firewall-Service – eine weitere Ebene der Absicherung.

Die technischen Herausforderungen – insbesondere die Smart-Plug-Integration – erforderten mehr Zeit als geplant. Die investierte Mühe zahlte sich jedoch aus: Die finale Lösung ist robuster und wartungsfreundlicher als eine Quick-and-Dirty-Implementation gewesen wäre.

4.2 Fazit

Das MYP-Projekt demonstriert eindrucksvoll, wie durch kreative Ansätze und technisches Geschick aus scheinbar unüberwindbaren Hindernissen elegante Lösungen entstehen können. Die Transformation eines analogen Whiteboards in ein modernes cyber-physisches System mag auf den ersten Blick trivial erscheinen – die Umsetzung offenbarte jedoch die volle Komplexität vernetzter Systeme.

Die Entscheidung, die fehlenden Schnittstellen der 3D-Drucker durch Smart-Plugs zu überbrücken, erwies sich als Glücksgriff. Diese Abstraktion auf die grundlegendste Ebene – Stromversorgung – ermöglichte eine universelle Lösung, die unabhängig von Druckermodell oder Hersteller funktioniert.

Die technische Exzellenz des Systems zeigt sich in den Details: Über 9.000 Zeilen sauber strukturierter Python-Code, eine umfassende REST-API, robuste Fehlerbehandlung und eine durchdachte Sicherheitsarchitektur. Der eigentliche Erfolg manifestiert sich jedoch in der

© 2025 - Till Tomczak

Praxistauglichkeit. Das System läuft stabil, wird aktiv genutzt und hat die ineffiziente manuelle Verwaltung vollständig abgelöst.

Persönlich stellte das Projekt eine intensive Lernerfahrung dar. Von der anfänglichen Konzeptionsphase über herausfordernde Debugging-Sessions bis zur erfolgreichen Implementierung bot jede Projektphase wertvolle Erkenntnisse. Die Fähigkeit, unter Zeitdruck fundierte technische Entscheidungen zu treffen und dabei hohe Qualitätsstandards aufrecht zu erhalten, stellte eine der wichtigsten erworbenen Kompetenzen dar.

4.3 Optimierungsmöglichkeiten

Das MYP-System bietet eine solide Basis für zukünftige Erweiterungen. Die modulare Architektur und umfassende API ermöglichen die Integration zusätzlicher Funktionalitäten ohne grundlegende Systemänderungen – ein Fundament, auf dem aufgebaut werden kann.

Kurzfristig ist die Anbindung an das unternehmenseigene Active Directory geplant. Die vorbereiteten Schnittstellen ermöglichen eine nahtlose Integration, sobald die erforderlichen Genehmigungen vorliegen. Diese Erweiterung würde die Benutzerverwaltung erheblich vereinfachen und die Akzeptanz im Unternehmensumfeld steigern.

Mittelfristig könnte bei Verfügbarkeit modernerer 3D-Drucker eine direkte Geräteintegration realisiert werden. Die Einbindung von OctoPrint oder vergleichbaren Systemen würde erweiterte Funktionen wie Druckfortschrittsüberwachung und Remote-Dateiverwaltung ermöglichen – Features, die das System auf ein neues Level heben würden.

Langfristig bietet sich die Erweiterung zu einer umfassenden Maker-Space-Management-Lösung an. Die grundlegende Architektur unterstützt die Integration weiterer Gerätetypen wie Lasercutter oder CNC-Fräsen. Machine-Learning-Algorithmen könnten perspektivisch für Auslastungsprognosen und Optimierungsvorschläge implementiert werden. Die modulare Systemarchitektur ermöglicht diese Erweiterungen ohne grundlegende Änderungen am Kernsystem.

4.4 Abnahme

ABNAHMEPROTOKOLL

Die formale Projektabnahme erfolgte am 2. Juni 2025 durch die Ausbildungsleitung der TBA. Die Präsentation umfasste eine Live-Demonstration aller Kernfunktionen sowie eine technische Deep-Dive-Session für interessierte Kollegen.

Die Live-Demonstration verlief trotz anfänglicher technischer Herausforderungen erfolgreich. Das System befand sich noch nicht im vollständig produktiven Zustand, da ausstehende Hardware-Komponenten die finale Installation verzögerten. Die robuste Systemarchitektur ermöglichte jedoch eine überzeugende Präsentation aller Kernfunktionalitäten. Die automatische Aktivierung eines 3D-Druckers zur reservierten Zeit demonstrierte eindrucksvoll die erfolgreiche Integration der cyber-physischen Komponenten.

Besonders positiv wurde die Wirtschaftlichkeit der Lösung bewertet. Mit Gesamtkosten unter 600 Euro (inklusive privat finanzierter Komponenten) liegt das System weit unter kommerziellen

© 2025 - Till Tomczak

Alternativen. Die Einsparungen durch automatisierte Abschaltung und optimierte Nutzung amortisieren die Investition binnen weniger Monate – ein Argument, das bei der Geschäftsführung Anklang fand.

Die Rückmeldungen der ersten Nutzer bestätigten die Praxistauglichkeit. Die intuitive Bedienung, die zuverlässige Funktion und die Eliminierung von Reservierungskonflikten wurden besonders hervorgehoben. Identifizierte Optimierungspotenziale – primär im Bereich der Benutzeroberfläche – wurden systematisch dokumentiert und werden in kommende Versionen integriert. Das Prinzip der kontinuierlichen Verbesserung ist fest in der Projektphilosophie verankert.

Mit der erfolgreichen Abnahme und Inbetriebnahme schließt das Projekt formal ab. Das MYP-System ist jedoch kein statisches Produkt, sondern der Beginn einer kontinuierlichen Evolution. Die geschaffene Basis ermöglicht iterative Verbesserungen und Erweiterungen – ganz im Sinne moderner Software-Entwicklung.

Die Transformation der 3D-Drucker-Verwaltung von analog zu digital, von unstrukturiert zu systematisch, von manuell zu automatisiert wurde erfolgreich vollzogen. Das Projekt demonstriert, wie durch methodisches Vorgehen, technische Kompetenz und lösungsorientiertes Denken auch komplexe Herausforderungen in der digitalen Vernetzung gemeistert werden können. Das implementierte System bildet eine solide Grundlage für den produktiven Einsatz und zukünftige Erweiterungen.

© 2025 - Till Tomczak

Anlagen

Übergabeprotokoll

© 2025 - Till Tomczak

© 2025 - Till Tomczak