**Einführung**

**Projektziel**

Ziel des Projekts ist es, eine Inventar-App zu entwickeln, die es ermöglicht, das gesamte Inventar einer Schule effizient zu verwalten. Nutzer können das Inventar durchsuchen, filtern und gezielt Informationen zu bestimmten Räumen, Schränken oder Gegenständen abrufen. Die App bietet außerdem die Möglichkeit, QR-Codes zu scannen, um direkt auf Inhalte zuzugreifen.

**Projektbegründung**

Die Verwaltung des Schulinventars stellt oft eine zeitaufwendige und fehleranfällige Aufgabe dar. Mit der digitalen Lösung soll die Organisation optimiert und der Zugriff auf Inventardaten vereinfacht werden. Lehrkräfte und Verantwortliche sparen Zeit, da die App eine schnelle Suche und Filterung ermöglicht.

**Projektschnittstellen**

* **Backend:** ASP.Net Core API mit EFCore zur Verwaltung der Datenbank. Beinhaltet Endpoints, welche im Frontend aufgerufen werden
* **Frontend:** Vue/Quasar-Web-App mit Tailwind CSS und TypeScript. Um Inventars-Daten anzuzeigen oder für Funktionen zum Hinzufügen oder Entfernen, werden Endpoints vom Backend aufgerufen.
* **Datenbank:** SQLite als leichtgewichtige, eingebettete Datenbank.

**Projektabgrenzung**

Die App fokussiert sich ausschließlich auf die Verwaltung des bestehenden Inventars. Funktionen wie Bestellmanagement, Reparaturübersicht oder automatisierte Benachrichtigungen bei Inventarveränderungen sind nicht Teil des Projekts. Ansprechpartner sind Mika Kuns und Jan Butzke (durchführende Entwickler), sowie der Lehrkörper Björn Beuthien, welcher die Aufgabenstellung gab.

**Planung**

**Projektphasen**

1. **Analyse und Planung:** Erstellung eines ER-Modells, einer Method-Auflistung und eines Techstacks. Prototyp des Programm Designs mit Hilfe von Figma erstellen.
2. **Entwicklung Backend:** Implementierung der Datenbank und Endpoints mittels EFCore und in einem ASP.Net Core Projekt.
3. **Entwicklung Frontend:** Aufbau der Benutzeroberfläche mit Vue und Quasar sowie Implementierung der QR-Code-Scan-Funktion.
4. **Integration:** Verknüpfung von Frontend und Backend, Testen der API-Aufrufe.
5. **Tests und Qualitätssicherung:** Durchführung umfassender Tests (u. a. Unit-Tests und End-to-End-Tests).
6. **Bereitstellung und Dokumentation:** Deployment der Web-App und Erstellung einer Benutzerdokumentation.

**Ressourcen**

* **Technisch:**
  + Entwicklungsumgebung für ASP.Net Core und Vue/Quasar.
  + SQLite-Datenbank.
  + Geräte für Tests (Smartphones, Tablets, Desktop-PCs).
* **Personell:** Zwei Entwickler (einer für Backend, einer für Frontend).
* **Zeitlich:** Ca. 80 Stunden Entwicklungszeit, aufgeteilt auf die oben genannten Phasen.

**Kosten**

Die Kosten für die Durchführung des Projekts belaufen sich auf 1530 €. Es wurde davon ausgegangen, dass die Kosten für einen Auszubildenden einschließlich Lohn, Personal, Geräte/Raum und Strom in einem Betrieb 15 € betragen. Die Kosten für einen Hochschulabsolventen belaufen sich auf 55 €, da ein höheres Gehalt sowie höhere Geräte- und Personalkosten anfallen.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Aufwand in Stunden** | **Kosten pro Stunde** | **Kosten** |
| Aufnahme Anforderung | 2 | 55€+15€ | 140 € |
| Konzept erstellen | 4 | 15€ | 60 € |
| Abnahme | 1 | 55€+15€ | 70 € |
| Entwicklung | 54 | 15€ | 810 € |
| Testen | 5 | 15€ | 75 € |
| Dokumentation | 11 | 15€ | 165 € |
| Ausliefern inkl. Endabnahme | 3 | 55€+15€ | 210 € |
|  | 80 |  | 1530 € |

**Wirtschaftlichkeitsanalyse**

Die digitale Inventar-App senkt den Zeitaufwand für die Inventarverwaltung erheblich, was langfristig Kosten einspart. Durch das Wissen, was wo zu finden ist, kann der Unterricht besser geplant werden und muss nicht innerhalb einer Unterrichtsstunde durch eine Suchaktion unterbrochen werden. Da bestehende Open-Source-Technologien (SQLite, Vue, Quasar) verwendet werden, bleiben die Entwicklungskosten niedrig. Ein zusätzlicher Vorteil ist die plattformübergreifende Nutzung, die keine spezifischen Hardwareanforderungen an die Schule stellt.

**Bewertung**

**Qualitätssicherung / Tests**

* **Unit-Tests:** Sicherstellen, dass zentrale Funktionen (z. B. Datenbankzugriff, API-Endpunkte) fehlerfrei arbeiten.
  + Wir haben mit NUnit unsere c# backend punkte getestet.  
    Das funktioniert, indem wir eine Datenbank dafür im Code mocken, und immer die eben eingefügten Daten abprüfen.  
    Wenn der status 200 kommt ist alles gut
* **Integrationstests:** Überprüfung der Kommunikation zwischen Frontend und Backend.
* **Usability-Tests:** Sicherstellung, dass die App auf verschiedenen Geräten einwandfrei bedient werden kann.
  + Wir haben unseren Browser bei der Entwiklung stehts zwischen Desktop und smartphone umgeschaltet, und so Getestet dass alles responsive ist. Außerdem nutzen wir keine Festen größen, sondern nur relative.
* **Fehlerbehandlung:** Validierung von Eingaben und Rückmeldungen bei unerwarteten Situationen (z. B. QR-Code-Fehler).
  + Wir haben händisch viele mögliche Falscheingaben getestet. Außerdem haben wir eine Katze über die Tastatur laufen lassen, ohne das eine Fehlermeldung erschien.
* **Performance-Tests:** Überprüfung der Ladezeiten und Effizienz der Filter- und Suchfunktionen.
  + Nach dem Firefox Profiler sieht alles Performant aus.