Selbständige Projektarbeit  
 Informatikmittelschule Frauenfeld

**Freezing Inventory**



Autor: Remo Thim  
Betreuer: Matthias Bolli

Inhalt

[1 Einleitung 4](#_Toc96639138)

[2 Informieren 4](#_Toc96639139)

[2.1 Ziele 4](#_Toc96639140)

[2.2 Zeitrahmen 4](#_Toc96639141)

[2.3 Technische Anforderungen 4](#_Toc96639142)

[3 Planen 4](#_Toc96639143)

[3.1 Plattform 4](#_Toc96639144)

[3.1.1 Betriebssystem 4](#_Toc96639145)

[3.1.2 App-Typ 5](#_Toc96639146)

[3.2 Funktionen 6](#_Toc96639147)

[3.2.1 Datenbank 6](#_Toc96639148)

[3.3 Framework 6](#_Toc96639149)

[3.3.1 Apache Cordova 6](#_Toc96639150)

[3.3.2 Flutter 7](#_Toc96639151)

[4 Entscheidung 7](#_Toc96639152)

[4.1 Beschreibung 7](#_Toc96639153)

[4.2 Umfang 7](#_Toc96639154)

[4.2.1 Must-Have 7](#_Toc96639155)

[4.2.2 Nice-to-Have 7](#_Toc96639156)

[4.3 Plattform und Technische Umsetzung 7](#_Toc96639157)

[4.4 Design 8](#_Toc96639158)

[4.4.1 Mock-Up 8](#_Toc96639159)

[4.4.2 Storyboard 8](#_Toc96639160)

[4.5 Datenbank 9](#_Toc96639161)

[4.5.1 ERD 10](#_Toc96639162)

[4.6 Veröffentlichen 10](#_Toc96639163)

[5 Realisieren 10](#_Toc96639164)

[5.1 GitHub 10](#_Toc96639165)

[5.2 JQuery 22 10](#_Toc96639166)

[5.3 Bootstrap + Datepicker 10](#_Toc96639167)

[5.4 Datenbank 11](#_Toc96639168)

[5.4.1 Öffnen 11](#_Toc96639169)

[5.4.2 SQL-Befehl ausführen 11](#_Toc96639170)

[5.5 Tabelle 11](#_Toc96639171)

[5.6 Kategorie auswählen 12](#_Toc96639172)

[5.7 Dropdownliste von Artikeln 12](#_Toc96639173)

[5.8 Artikel verbrauchen/löschen 13](#_Toc96639174)

[5.9 Artikel hinzufügen 14](#_Toc96639175)

[5.10 Neue Kategorie 14](#_Toc96639176)

[5.11 Artikel bearbeiten 15](#_Toc96639177)

[6 Kontrollieren / Testing 16](#_Toc96639178)

[6.1 Blackbox-Testing 16](#_Toc96639179)

[7 Auswerten 19](#_Toc96639180)

[7.1 Erreichen der Ziele 19](#_Toc96639181)

[7.1.1 Must-Have 19](#_Toc96639182)

[7.1.2 Nice-to-Have 20](#_Toc96639183)

[7.2 Known Bugs/Problems 20](#_Toc96639184)

[7.3 Probleme 20](#_Toc96639185)

[7.3.1 Zeit 20](#_Toc96639186)

[7.4 Cordova 20](#_Toc96639187)

[7.5 Fazit 20](#_Toc96639188)

[8 Anleitungen 20](#_Toc96639189)

[8.1 Installation 20](#_Toc96639190)

[8.2 Benutzeranleitung 20](#_Toc96639191)

[9 Abbildungsverzeichnis 21](#_Toc96639192)

# Einleitung

Dies ist die Dokumentation zur Semesterarbeit. Die Dokumentation beschreibt den gesamten Prozess bis zur Fertigstellung einer App und ist nach dem IPERKA-Modell gegliedert.

# Informieren

## Ziele

Ziel der Semesterarbeit ist es, eine App zu entwickeln, welche die Verwaltung der Produkte innerhalb eines Kühl- und/oder Gefrierschranks übernehmen soll. Diese App soll wie bereits erwähnt auf Android funktionieren.

## Zeitrahmen

Die App soll bis zum 25. Februar 2022/17:00 Uhr fertiggestellt werden. Insgesamt sollen ca. 80 Arbeitsstunden in das Projekt fliessen.

## Technische Anforderungen

Folgende Anforderungen sollen erfüllt werden:

* Nutzung einer Datenbank
* Einfache Bedienung durch den Endnutzer

Gegebene Ziele sollen auf irgendeine Art erreicht werden, d.h. es existieren keine Vorgaben seitens des Auftraggebers.

# Planen

Das folgende Kapitel handelt von all den verschiedenen Lösungsmöglichkeiten für das Erreichen der technischen Anforderungen.

## Plattform

### Betriebssystem

#### Android

Android ist ein von Google entwickeltes Betriebssystem welches circa 71.8% des Marktanteils weltweit ausmacht. Damit ist es das am weitesten verbreitete Betriebssystem. Die Entwicklung für Android gestaltet sich sehr einfach, da schnell und unkompliziert Installationsdateien erstellt und anschliessend auf einem Gerät installiert und getestet werden können.

Abbildung 1 Android Logo

Vorteile:

* weltweit genutzt
* Hardware ist uneingeschränkt nutzbar
* einfache Entwicklung, Testing und Verbreitung

Nachteile:

* Hardware variiert stark

#### IOS

Das Betriebssystem IOS ist das zweitmeist verbreitete Betriebssystem in der Welt. Der Marktanteil des Betriebssystems beträgt circa 27.2%. Anders als Android, welches auf ziemlich allen mobilen Endgeräten läuft, funktioniert IOS grundsätzlich nur auf Apple Geräten.

Abbildung 2 IOS Logo

Vorteile:

* Zugriff auf viele spezifische Funktionen
* IOS-Ökosystem

Nachteile:

* Entwicklung läuft nur über ein Apple Gerät und deren Programmiersprache Swift
* Veröffentlichung nur nach Testing seitens Apple

### App-Typ

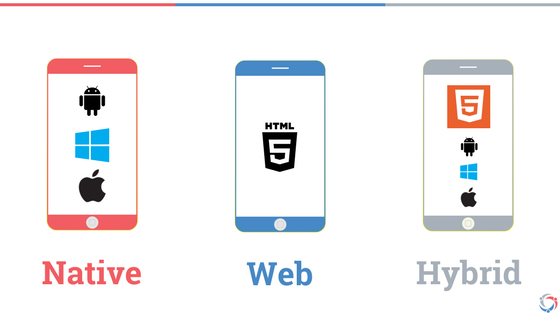
Es gibt drei Arten, wie man eine App entwickeln kann. Das folgende Kapitel behandelt alle Arten sowie deren Vor- und Nachteile.

Abbildung 3 Native

#### Native

Native Apps werden spezifisch für ein Betriebssystem entwickelt. Für alle nativen Android Applikationen wäre die Sprache «Java», und für alle Apple-nativen Applikationen wäre es Swift und SwiftUI.

Vorteile:

* Potenzial eines Gerätes vollständig nutzbar
* UI ist relativ normalisiert

Nachteile:

* falls die App auf anderen Plattformen verfügbar sein soll, muss diese von Null auf entwickelt werden.

#### Hybrid

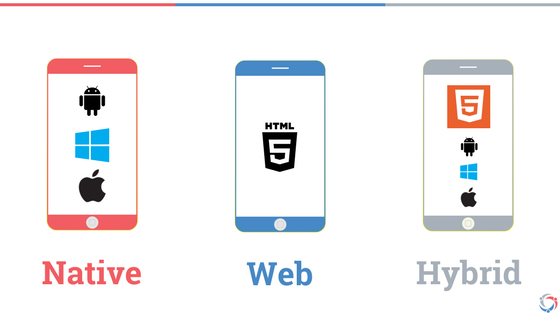
Bei einer Hybrid-App wird mit bewährten Webtechnologien entwickelt und anschliessend durch einen Webviewcontainer in eine native App gepackt.

Abbildung 4 Webapp

Vorteile:

* Cross-platform-tauglich
* vielseitig
* günstiger als Entwicklung von mehreren nativen Apps

Nachteile:

* kann nicht volles Potenzial der Hardware ausschöpfen

#### Crossplatform

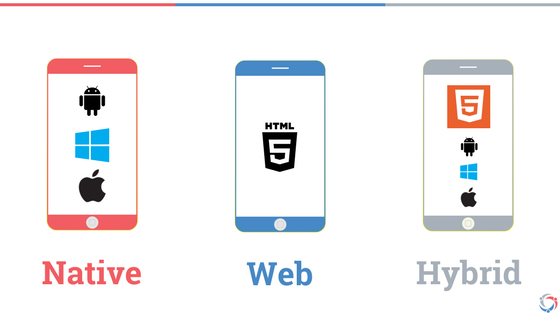
Eine Crossplatform-App wird mit nicht für ein plattform-spezifisches Betriebssystem entwickelt. Schlussendlich wird die App dann auf die spezifischen Betriebssysteme portiert.

Abbildung 5 Hybrid

Vorteile:

* weniger Arbeit als Entwicklung für Plattformen separat
* Frameworks können durch Plugins auch native Funktionen bieten

Nachteile:

* Funktionen können auf sich warten lassen
* forderndes Entwickeln
* kein Zugriff auf vereinzelte native Funktionen

## Funktionen

### Datenbank

Damit der User überhaupt die ganzen Daten speichern kann, ist eine Datenbank vonnöten.

#### Firebase

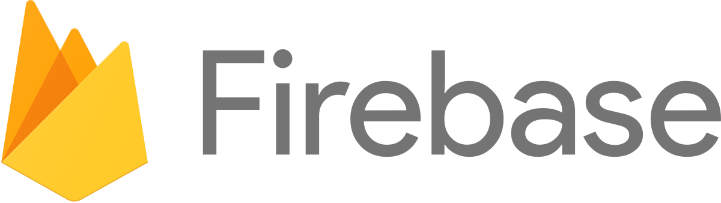
Firebase ist eine Datenbank mit hierarchischer Struktur. Sie wird von Google entwickelt und auch betrieben. Die Datenbank läuft über die Google Server und ist damit allzeit, sofern eine Internetverbindung besteht, erreichbar. Die Bedienung von Firebase ist leicht, da es eine JSON-gefüllte Datenbank ist.

Abbildung 6 Firebase Logo

#### SQLite

Ermöglicht kleine, lokale, relationale Datenbanken für Mobilgeräte zu entwickeln. Die Datenbank befindet sich lokal auf dem Gerät und kann somit auch ohne Internetverbindung genutzt werden.

Abbildung 7 SQLite Logo

## Framework

Damit man eine Hybride-App entwickeln kann, braucht man ein Framework. Nachfolgend werden die beiden beliebtesten Frameworks erklärt. Es wird auf den Vergleich mit nativen und Crossplatform verzichtet, da diese nicht im Rahmen des Möglichen für mich standen.

### Apache Cordova

Ein Bild, das Text, Uhr enthält.

Automatisch generierte BeschreibungCordova ist ein Framework, welches einem ermöglicht mit den gängigen Webentwicklungssprachen HTML 5, CSS 3 und JavaScript Hybride Apps zu entwickeln. Cordova ermöglicht weiterführend durch die Benutzung eines Plugins SQLite in die App zu integrieren. Erfahrung mit Cordova ermöglicht ebenfalls die schnelle Einarbeitung in die Materie der hybriden App-Entwicklung.

Abbildung 8 Cordova Logo

### Flutter

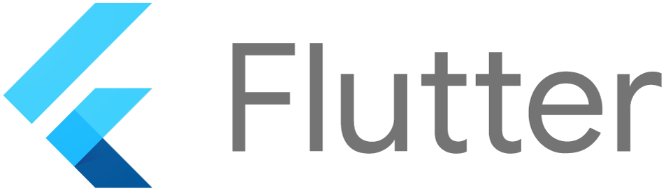
Flutter ist eine von Google entwickeltes Framework, welches einem ebenfalls ermöglicht, hybride Apps zu entwickeln. Weiterführend benutzt Flutter jedoch C++ und Dart als Programmiersprachen, womit ich keine Erfahrung habe. Andererseits wird ebenfalls SQLite als Datenbank unterstützt.

Abbildung 9 Flutter Logo

# Entscheidung

## Beschreibung

«Freezing Inventory» ist eine Android App, welche die Lagerverwaltung für den Gefrierschrank und oder Kühlschrank übernimmt. Mit dieser App wird dafür gesorgt, dass möglichst keine Produkte ablaufen und der Überblick behalten werden kann, welche Produkte sich gerade darin befinden. Weiterführend dient es als indirekte Einkaufsliste, da man sieht, was nicht mehr vorhanden ist.

## Umfang

### Must-Have

* Neues Produkt der Datenbank hinzufügen
* Produkte aus Datenbank entfernen (Gebrauch)
* Produkte in benutzerdefinierbare Kategorien einteilen
* Benachrichtigung bei eventuellem Ablauf eines Produktes

### Nice-to-Have

* Kamera als Barcodescanner zur schnelleren Verwaltung

## Plattform und Technische Umsetzung

Ich besitze einen Windows-Rechner, was bedeutet, die Entwicklung einer nativen IOS App entfällt somit gänzlich. Daher habe ich mich dazu entschieden, eine hybride Android-App zu entwickeln. Diese würde auf Basis von Cordova entwickelt werden, da ich mit Cordova schon Erfahrung im ÜK gesammelt habe, und ich gerne mit den Webtechnologien wie HTML, CSS und JavaScript arbeite.

## Design

### Ein Bild, das Tisch enthält. Automatisch generierte BeschreibungMock-Up

Abbildung 10 Mock-Up

Das Grunddesign besteht daraus, dass eine grosse Tabelle den Grossteil der Bildschirmfläche einnimmt, um die Übersicht von vielen Artikeln zu vereinfachen. Die App wird keine Querformatunterstützung besitzen, da es für diese Art von Applikation keinen Sinn macht.

### Storyboard

Im folgenden Storyboard wird erläutert, was all die einzelnen Unterseiten der App machen.

#### Inventar

Auf der ersten Seite namens «Inventar» kann der Nutzer all seine Produkte sehen, welche er in der Datenbank gespeichert hat. Ebenso kann er auf dieser Seite nach Kategorien sortieren.

#### Verbrauch

Auf dieser Seite kann man im Dropdown Menu das Produkt auswählen, welches man aufbrauchen/entnehmen möchte. Anschliessend gibt man ein, wie viel von diesem Produkt verbraucht werden soll (Anzahl), und bestätigt das dann mit einem Klick auf «Verbrauchen».

#### Hinzufügen

Auf dieser Unterseite kann man einen neuen Artikel hinzufügen. Man wählt ganz simpel einen Namen, die Anzahl, ein Ablaufdatum und eine Kategorie. Hinzufügen macht man dann, indem man auf den Knopf «Hinzufügen» klickt.

#### Bearbeiten

Auf dieser Seite wählt man wie beim Verbrauchen ein Produkt aus, welches man Überarbeiten möchte, und ändert anschliessend die Daten über Inputfelder. Bestätigen wird erledigt, indem man auf «Speichern» drückt.

#### Löschen

Analoges Prinzip wie beim Verbrauchen.

Abbildung 11 Storyboard



## Datenbank

Bezüglich der Datenbank habe ich mich für SQLite entschieden, da diese mit ihrer relationalen Natur ermöglicht, feste Kategorien mit einzelnen Artikeln zu verbinden. Ebenso ist SQLite eine lokale Datenbank, weswegen diese App folglich auch dann benutzt werden kann, wenn gerade einmal kein Internet zur Verfügung steht. Hinzu kommt, dass Cordova mittels eines Plugins besagtes SQLite unterstützt.

### ERD

Abbildung 12 ERD

## Veröffentlichen

Eine Google Playstore Veröffentlichung der App ist nicht vorgesehen, da dies mit Kostenaufwand verbunden sind.

# Realisieren

## GitHub

GitHub dient im Falle dieser Projektarbeit als Cloud Speichermedium, um dafür zu sorgen dass, falls meine Daten auf dem Computer verloren gehen, ich diese einfach wieder vom Repository klonen kann. Diese Entscheidung wurde gefällt, da ich dieses Vorgehen bereits für mein ÜK Projekt benutzt und nur Positives erlebt habe.

Abbildung 13 GitHub Logo

Um die Daten zwischen dem Repository und dem Computer hin- und herzuschieben, benutze ich das Programm «GitHub Desktop». Ebenso arbeite ich mit «Visual Studio Code», was eine Integrierte Klon wie auch Synchronisation mit dem Repository ermöglicht.

## JQuery 22

Um einem die Möglichkeit zu geben zwischen den Unterseiten zu wechseln, benötigt es JQuery welches beim Wechsel die richtigen HTML- und JS-Dateien in ein DIV vom «Index.html» lädt. Folgender Code implementiert dies:

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

## Bootstrap + Datepicker

Damit ich eine schöne App habe, wollte ich auf Bootstrap zurückgreifen, da wir mit diesem schon sehr lange arbeiten. Ebenso musste ich per speziellem Bootstrap link mir den dazugehörigen Datumswähler ziehen.

<script type="text/javascript"

src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.4.1/js/bootstrap-datepicker.min.js"></script>

<link rel="stylesheet"

href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.4.1/css/bootstrap-datepicker3.css" />

## Datenbank

Im folgenden Unterkapitel wird die Kommunikation mit der SQLite Datenbank erläutert. Zuvor möchte ich ein grosses Dankeschön an folgenden Stack Overflow Thread aussprechen, welcher mir viel geholfen hat zu verstehen, wie Sqlite mit Cordova funktioniert: [Hyperlink](https://stackoverflow.com/questions/26604952/a-simple-cordova-android-example-including-sqlite-read-write-and-search/26609184)

### Öffnen

Damit man eine Aktion auslösen kann, muss man mit folgendem Code die Datenbank öffnen/erstellen:

var db = window.openDatabase("Inventory", "1.0", "Inventory", 200000);

### SQL-Befehl ausführen

Damit man SQL-Befehle auf der Datenbank ausführen, kann muss folgendes machen:

db.transaction(populateDB, errorCB);

Dieser Code-Teil gibt eine spezielle Variable an «populateDB» weiter. Der «errorCB» ist nur dafür da, dass, falls ein Fehler bei einem SQL-befehl passiert, ein HTML-Alert angezeigt wird mit dem Fehlercode. Nachstehend die Funktion «populateDB»:

function populateDB(tx) {

tx.executeSql('CREATE TABLE IF NOT EXISTS TKategorie (KatId INTEGER PRIMARY KEY AUTOINCREMENT, KatName)');

tx.executeSql('CREATE TABLE IF NOT EXISTS TArtikel (ArtId INTEGER PRIMARY KEY AUTOINCREMENT,KatId, ArtName,ArtAnz,ArtAblaufdatum)');

}

Die Variable «tx» steht für die oben erwähnte spezielle Variable. Wie man nun erkennen kann, wird folglich mit «tx.executeSql('SQL-Befehl')» besagter Befehl auf der Datenbank ausgeführt.

## Tabelle

Um die Tabelle basierend auf einem SQL-Select zu erstellen, muss folgendes geschehen.

Datenbank öffnen und Funktion «queryDB» aufrufen:

db.transaction( queryDB, errorCB );

Den SQL-Befehl ausführen und Resultate an «querySuccess» weitergeben.

function queryDB(tx) {

tx.executeSql("SELECT ArtName,ArtAnz,ArtId,strftime('%d.%m.%Y',ArtAblaufdatum) as ptime FROM TArtikel ", [], querySuccess, errorCB);

Die Resultate durch eine For-Schleife laufen lassen und anschliessend als HTML-Code in das dazugehörige Div laden:

function querySuccess(tx, results) {

var tblText = '<table class="table table-striped table-bordered"><thead><tr><th>Name</th> <th>Anzahl</th> <th>Ablaufdatum</th></tr></thead><tbody>';

var len = results.rows.length;

for (var i = 0; i < len; i++) {

tblText += '<tr><td>' + results.rows.item(i).ArtName + '</td><td>' + results.rows.item(i).ArtAnz + '</td><td>' + results.rows.item(i).ptime + '</td></tr>';

}

tblText += "</tbody></table>";

document.getElementById("table").innerHTML = tblText;

}

## Kategorie auswählen

Damit man auf der Inventarseite eine Kategorie auswählen kann und sich anschliessend die Tabelle anpasst, sind folgende Schritte notwendig:

Bei Änderung von Dropdownliste schauen, welchen Value die ausgewählte Kategorie hat (entspricht der KatId). Anschliessend prüfen, ob value = 0 (entspricht der Kategorie: Alle) oder eine andere Zahl ist und schliesslich den Select-Befehl anpassen. Schlussendlich per «KatAnzeige» den SQL-Befehl einleiten:

function KatWahl(selectObject, tx) {

var Auswahl = selectObject.value;

if (Auswahl == 0) {

select = "SELECT ArtName,ArtAnz,strftime('%d.%m.%Y',ArtAblaufdatum) as ptime FROM TArtikel";

} else {

select = "SELECT ArtName,ArtAnz,strftime('%d.%m.%Y',ArtAblaufdatum) as ptime FROM TArtikel where KatId = " + Auswahl

}

db.transaction(KatAnzeige, errorCB)

}

Einen SQL-Befehl ausführen basierend auf der Variable namens «select». Das Ergebnis wird folglich an «querySuccess» weitergegeben und in der Funktion in eine Tabelle umgewandelt (Verweis: 5.5 Letzter Code abschnitt):

function KatAnzeige(tx) {

tx.executeSql(select, [], querySuccess, errorCB);

}

## Dropdownliste von Artikeln

Damit der Nutzer die richtigen Artikel überhaupt verbrauchen kann, dachte ich mir, dass eine Dropdownliste basierend auf der Tabelle TArtikel das Beste wäre. Dies funktioniert wie folgt:

Anfangs wird ein Select-Befehl ausgeführt und die Resultate an die Funktion «FillBenuntzung» gesendet:

function Benutzung(tx) {

tx.executeSql("SELECT ArtName,ArtId,strftime('%d.%m.%Y',ArtAblaufdatum) as ptime FROM TArtikel", [], fillBenutzung, errorCB);

}

Danach wird wie bei der Tabelle Erstellung ein HTML-String kreiert, welcher schlussendlich in das DIV geladen wird. Die einzelnen Optionen innerhalb des Dropdowns werden durch eine For-Schleife an den String angefügt. Damit man abfragen kann, welche Option ausgewählt ist, erhalten die einzelnen Optionen als Wert die ArtikelId:

function fillBenutzung(tx, results) {

var VerHtml = '<label><b>Artikel:</b></label><br><select id="select" class="form-select">'

var len = results.rows.length;

for (var i = 0; i < len; i++) {

VerHtml += "<option value=" + results.rows.item(i).ArtId + ">" + results.rows.item(i).ArtName + " vom " + results.rows.item(i).ptime + "</option>";

}

VerHtml += '</select><br><label><b>Anzahl:</b></label><br><input type="number" id="Anzahl" class="form-control w-25" min="1"><button id="VerbButton" class="btn btn-light" onclick="Verbauch()">Verbrauchen</button>'

document.getElementById("verbrauch").innerHTML = VerHtml;

}

## Artikel verbrauchen/löschen

Damit der Nutzer auch die einzelnen Produkte aufbrauchen kann, benötigt es folgenden Code:

Wenn der «Verbrauch» Button gedrückt wird, öffnet es die Datenbank und es wird geprüft, was für ein Wert im Feld Anzahl (number input field) ist. Wenn die Zahl grösser gleich 1 ist, wird der SQL-Befehl eingeleitet, andererseits wird ein Alert (falls Zahl kleiner 1 ist) angezeigt.

function Verbauch() {

var db = window.openDatabase("Inventory", "1.0", "Inventory", 200000);

var wert = document.getElementById("Anzahl");

if (wert.value < 1) {

alert("Die Anzahl muss grössergleich 1 sein!")

} else {

db.transaction(VerbrauchSQL, errorCB);

}

}

In der folgenden Funktion wird dann geprüft, welchen Wert der ausgewählte Artikel hat. Anschliessend wird die Anzahl, die verbraucht werden soll, aus dem «Anzahl» Feld gezogen und der SQL-Befehl ausgeführt, In welchem die ArtAnz um **den** Wert, welcher im Anzahlfeld steht, reduziert wird.

function VerbrauchSQL(tx) {

var auswahl = document.getElementById("select");

var wert = document.getElementById("Anzahl")

tx.executeSql("UPDATE TArtikel set ArtAnz = ArtAnz - " + wert.value + " where ArtId = " + auswahl.value, [], queryDB, errorCB);

}

Bei der vorherigen Funktion wird nach der Reduzierung die Tabelle neu kreiert. Hierbei wird aber im «querysuccess» geprüft, ob die Anzahl eines Artikels unter 1 ist und anschliessend in den folgenden Funktionen gelöscht.

function deleteRow(ID) {

row = ID;

var db = window.openDatabase("Inventory", "1.0", "Inventory", 200000);

db.transaction(deleteSelect, errorCB);

}

function deleteSelect(tx) {

tx.executeSql('DELETE FROM TArtikel WHERE ArtId = ' + row, [], queryDB, errorCB)

db.transaction(Benutzung, errorCB);

}

## Artikel hinzufügen

In der folgenden Funktion können Artikel hinzugefügt werden.

Das Hinzufügen funktioniert, indem der Nutzer eine Kategorie auswählt (Dropdown Verweis: 5.6), und in die drei Input-Felder die notwendigen Daten einfügt. Diese werden dann kontrolliert, ob diese grösser gleich 1 und nicht leer sind. Diese Funktion hier abzubilden, führt wegen des grossen Umfangs zu weit, weshalb ich darauf verzichte (kann auf Wunsch nachgereicht werden).

In der folgenden Funktion wird jetzt ein Insert SQL-Befehl ausgeführt mit den Werten aus den Input-Feldern. Ebenso werden die Input-Felder geleert und ein Alert abgegeben, daas der Nutzer weiss, dass es funktioniert hat.

function hinzufuegen(tx) {

tx.executeSql("INSERT INTO TArtikel (KatID,ArtName,ArtAnz,ArtAblaufdatum) VALUES (" + document.getElementById("select").value + ",'" + document.getElementById("HinzName").value + "','" + document.getElementById("HinzAnz").value + "','" + document.getElementById("HinzDatum").value + "')", [], queryDB, errorCB);

document.getElementById("select").value = 0;

document.getElementById("HinzName").value = "";

document.getElementById("HinzAnz").value = "";

document.getElementById("HinzDatum").value = "";

alert("Artikel Hinzugefügt!");

}

## Neue Kategorie

Damit der Nutzer eine neue benutzerdefinierte Kategorien hinzufügen kann, wird folgendes gemacht:

Wenn in der Dropdownliste «neue Kategorie» ausgewählt wird, ändert es das Aussehen der «Hinzufügen»-Seite, indem es nur noch ein Input-Feld hat.

function changeHtml() {

if (document.getElementById("select").value == -1) {

document.getElementById("Hinzufuegen").style.display = "none";

document.getElementById("newKat").style.display = "block";

} else {

document.getElementById("newKat").style.display = "none";

document.getElementById("Hinzufuegen").style.display = "block";

}

}

Danach wird, wenn der Hinzufügen-Knopf gedrückt wird, kontrolliert, ob das Input-Feld für den neuen Namen leer ist oder nicht. Anschliessend wird typischerweise die Datenbank geöffnet und der SQL-Befehl eingeleitet:

function NewKat() {

if (document.getElementById("neueKat").value != "") {

var db = window.openDatabase("Inventory", "1.0", "Inventory", 200000);

db.transaction(newKatSQL, errorCB);

} else {

alert("Bitte Name für neue Kategorie eingeben!")

}

}

In folgender Funktion wird der benötigte Insert-Befehl für die neue Kategorie ausgeführt und anschliessend die Dropdownliste neu kreiert.

function newKatSQL(tx) {

tx.executeSql('INSERT INTO TKategorie (KatName) VALUES ("'+document.getElementById("neueKat").value+'")',[],dropdown, errorCB)

}

## Artikel bearbeiten

Damit der Nutzer seine Artikel bearbeiten kann, benötigt es folgende Funktionen:

Es wird ein einfacher Select-Befehl ausgeführt, welcher auf einem Dropdownlisten-Wert (analog zu 5.7) basiert:

function FillBearbeiten(tx) {

var selected = document.getElementById("select");

tx.executeSql('select \* from TArtikel WHERE ArtId = ' + selected.value, [], queryBearbeiten, errorCB)

}

Daraufhin wird auf Basis der Resultate aus dem Select ein HTML-Code kreiert, welcher schlussendlich in das dazugehörige DIV lädt. In diesem HTML-Code werden Input-Felder erstellt, welche mit den Werten aus dem Select gefüllt werden.

function queryBearbeiten(tx, results) {

var BearbeitenHTML = '<label>Name:</label><input type="text" id="BearName" class="form-control" value="' + results.rows.item(0).ArtName + '"><br>'

BearbeitenHTML += '<label>Anzahl:</label><input type="number" id="BearAnzahl" class="form-control" value="' + results.rows.item(0).ArtAnz + '"><br>'

BearbeitenHTML += '<label>Ablaufdatum:</label><input type="date" id="BearDatum" value="' + results.rows.item(0).ArtAblaufdatum + '"><br>'

BearbeitenHTML += '<button id="VerbButton" class="btn btn-light" onclick="submit()">Speichern</button>'

document.getElementById("Bearbeiten").innerHTML = BearbeitenHTML;

}

Der Nutzer verändert dann die Werte in den Input-Feldern. Anschliessend, wenn der Button «Speichern» gedrückt wird, wird ein einfacher Update SQL-Befehl eingeleitet. Ebenso wird, nachdem der Artikel bearbeitet wurde, die Dropdownliste neu erstellt.

function submit() {

var db = window.openDatabase("Inventory", "1.0", "Inventory", 200000);

db.transaction(updateTable, errorCB);

db.transaction(select, errorCB);

}

In der folgenden Funktion wird dann der Update-SQL-Befehl ausgeführt und die Tabellenneuerstellung eingeleitet.

function updateTable(tx) {

tx.executeSql("UPDATE TArtikel set ArtName = '" + document.getElementById("BearName").value + "', ArtAnz = '" + document.getElementById("BearAnzahl").value + "', ArtAblaufdatum = '" + document.getElementById("BearDatum").value + "' where ArtId = " + document.getElementById("select").value, [], queryDB, errorCB)

}

# Kontrollieren / Testing

## Blackbox-Testing

Blackbox-Test: Freezing Inventory

Tester: Lorenz Hoffmann

Programm Version: 1.0.0

Betriebssystem: Android-Version 11

Datum: 24.2.2022

|  |  |  |
| --- | --- | --- |
| TESTING | ERGEBNISS | BESTANDEN Y/N |
| Öffnen sie die App | App öffnet ohne Probleme | Y |
| Klicken sie auf den «Verbrauchen» Tab | Die Unterseite «Verbrauchen» öffnet sich. | Y |
| Klicken sie auf den «Übersicht» Tab | Die Unterseite «Übersicht» öffnet sich. | Y |
| Klicken sie auf den «Verwalten» Tab | Die Unterseite «Verwalten (hinzufügen)» öffnet sich. | Y |
| Klicken Sie auf den «Verwalten» Tab und anschliessend auf den «Bearbeiten» Tab | Die Unterseite unter «Verwaltung» namens «Bearbeiten» öffnet sich. (Error: «ERROR PROCESSING SQL: undefinded», aber es öffnet sich trotzdem) | N |
| KlickenSie auf den «Verwalten» Tab und anschliessend auf den «Hinzufügen» Tab | Die Unterseite unter «Verwaltung» namens «Hinzufügen» öffnet sich. | Y |
| Klicken sie auf den «Verwalten» Tab und anschliessend auf den «Löschen» Tab | Die Unterseite unter «Verwaltung» namens «Löschen» öffnet sich. | Y |
| Klicken Sie im «Verwalten» Tab auf «Hinzufügen». Anschliessend klicken Sie auf die Dropdownliste und wählen «neue Kategorie» | Der Tab «Verwalten -> Hinzufügen» ändert sich und zeigt ein Eingabefeld mit dem Titel «Kategoriename», der Dropdownliste darüber und einem Button mit der Beschriftung «Hinzufügen» an | Y |
| Klicken Sie im «Verwalten» Tab auf «Hinzufügen». Anschliessend klicken Sie auf die Dropdownliste und wählen «neue Kategorie». Nun geben Sie im Eingabefeld «Kategoriename» «Fisch» ein und klicken auf hinzufügen. | Wenn sie nun wieder auf die Dropdownliste klicken sollte eine der Auswahl Möglichkeiten «Fisch» sein. | Y |
| Klicken Sie im «Verwalten» Tab auf «Hinzufügen». Anschliessend Klicken sie auf die Dropdownliste und wählen «Fisch» aus. Klicken Sie nun auf Hinzufügen. | Die App gibt einen «Alert» ab, welcher den Nutzer dazu auffordert einen «Namen» einzugeben. | Y |
| Klicken Sie im «Verwalten» Tab auf «Hinzufügen». Anschliessend klicken Sie auf die Dropdownliste und wählen «Fisch» aus. Geben Sie Im Eingabefeld «Name» «Fischstäbchen» ein. Klicken Sie nun auf Hinzufügen. | Die App gibt einen «Alert» ab, welcher den Nutzer dazu auffordert eine «Anzahl» einzugeben. | y |
| Klicken sie im «Verwalten» Tab auf «Hinzufügen». Anschliessend klicken Sie auf die Dropdownliste und wählen «Fisch» aus. Geben Sie Im Eingabefeld «Name» «Fischstäbchen» und im Feld «Anzahl» «2» ein. Klicken Sie nun auf Hinzufügen. | Die App gibt einen «Alert» ab, welcher den Nutzer dazu auffordert, ein «Ablaufdatum» einzugeben. | Y |
| Klicken Sie im «Verwalten» Tab auf «Hinzufügen». Anschliessend klicken Sie auf die Dropdownliste und wählen «Fisch» aus. Geben Sie Im Eingabefeld «Name» «Fischstäbchen», im Feld «Anzahl» «2» und wählen Sie ein Ablaufdatum aus, indem Sie auf das Feld klicken ein. Klicken Sie nun auf Hinzufügen. | Es sollte ein Alert erscheinen welcher Bestätigt das es hinzugefügt wurde. Ebenso sollte in der Tabelle auf der gleichen Seite nun der gerade hinzugefügte Artikel erscheinen. | Y |
| Klicken Sie im «Verwalten» Tab auf «Hinzufügen». Anschliessend klicken Sie auf die Dropdownliste und wählen «Fisch» aus. Geben sie Im Eingabefeld «Name» «Backfisch», im Feld «Anzahl» «2» und wählen Sie ein Ablaufdatum aus, indem Sie auf das Feld klicken ein. Klicken Sie nun auf Hinzufügen. | Es sollte ein Alert erscheinen, welcher bestätigt, dass es hinzugefügt wurde. Ebenso sollte in der Tabelle auf der gleichen Seite nun der gerade hinzugefügte Artikel erscheinen. | Y |
| Klicken sie im «Verwalten» Tab auf «Bearbeiten». Anschliessend klicken Sie auf die Dropdownliste und wählen «Fischstäbchen vom …» aus. | Die Eingabefelder unterhalb der Dropdownliste sollten sich mit den Daten des Produktes füllen. | y |
| Klicken Sie im «Verwalten» Tab auf «Bearbeiten». Anschliessend klicken Sie auf die Dropdownliste und wählen «Fischstäbchen vom …» aus. | Die Eingabefelder unterhalb der Dropdownliste sollten sich von den Daten der «Fischstäbchen» zu den Daten des «Backfischs» ändern. | y |
| Klicken Sie im «Verwalten» Tab auf «Bearbeiten». Anschliessend klicken Sie auf die Dropdownliste und wählen «Fischstäbchen vom …» aus. Anschliessend ändern Sie den Namen des Artikels im Eingabefeld «Name» zu «Iglo Fischstäbchen». Klicken Sie nun auf «Speichern». | In der Tabelle unterhalb der Eingabefelder sollte sich der Name des Produktes auf «Iglo Fischstäbchen». ändern. | Y |
| Klicken Sie im «Verwalten» Tab auf «Bearbeiten». Anschliessend klicken Sie auf die Dropdownliste und wählen «Iglo Fischstäbchen vom …» aus. Anschliessend ändern Sie die Anzahl des Artikels im Eingabefeld «Anzahl» zu «3». Klicken Sie nun auf Speichern. | In der Tabelle unterhalb der Eingabefelder sollte sich die Anzahl des Produkts auf «3» ändern. | Y |
| Klicken Sie im «Verwalten» Tab auf «Bearbeiten». Anschliessend klicken Sie auf die Dropdownliste und wählen «Iglo Fischstäbchen vom …» aus. Anschliessend ändern Sie das Ablaufdatum des Artikels im Eingabefeld «Ablaufdatum» zum «1.1.2022». Klicken Sie nun auf «Speichern». | In der Tabelle unterhalb der Eingabefelder sollte sich das Ablaufdatum des Produkts auf «1.1.2022» ändern. | y |
| Klicken sie im «Verwalten» Tab auf «Löschen». Anschliessend klicken Sie auf die Dropdownliste und wählen «Iglo Fischstäbchen vom …» aus. Anschliessend klicken Sie auf «Löschen». | Ein Alert sollte erscheinen, welcher dem Nutzer sagt, dass die «Anzahl» «grösser/gleich 1» sein muss. | Y |
| Klicken sie im «Verwalten» Tab auf «Löschen». Anschliessend klicken Sie auf die Dropdownliste und wählen «Iglo Fischstäbchen vom …» aus. Geben Sie anschliessend im «Anzahl» Feld «1» ein. Klicken sie auf «Löschen». | Die «Iglo Fischstäbchen» Anzahl sollte sich auf 2 reduzieren. | Y |
| Klicken Sie im «Verwalten» Tab auf «Löschen». Anschliessend klicken Sie auf die Dropdownliste und wählen «Iglo Fischstäbchen vom …» aus. Geben Sie anschliessend im «Anzahl» Feld «2» ein. Klicken Sie auf «Löschen». | Die «Iglo Fischstäbchen» sollten aus der Tabelle verschwinden. | Y |
| Klicken Sie auf den Knopf «Verbrauchen». Wählen Sie «Backfisch vom …» aus. Geben Sie anschliessend im «Anzahl» Feld «1» ein. Klicken sie auf «Verbrauchen». | Der «Backfisch» sollte sich auf 1 reduzieren. | Y |
| Klicken Sie im «Verwalten» Tab auf «Hinzufügen». Anschliessend klicken Sie auf die Dropdownliste und wählen «neue Kategorie». Nun geben Sie im Eingabefeld «Kategoriename» «Fleisch» ein und klicken auf «Hinzufügen». | Wenn Sie nun wieder auf die Dropdownliste klicken, sollte eine der Auswahlmöglichkeiten «Fleisch» sein. | Y |
| Klicken Sie im «Verwalten» Tab auf «Hinzufügen». Anschliessend klicken Sie auf die Dropdownliste und wählen «Fleisch» aus. Geben Sie im Eingabefeld «Name» «Rind», im Feld «Anzahl» «2» und wählen Sie ein Ablaufdatum aus, indem Sie auf das Feld klicken ein. Klicken Sie nun auf «Hinzufügen». | Es sollte ein Alert erscheinen, welcher bestätigt, dass es hinzugefügt wurde. Ebenso sollte in der Tabelle auf der gleichen Seite nun der gerade hinzugefügte Artikel erscheinen. | Y |
| Klicken Sie auf «Übersicht». Wählen Sie im Dropdown «Fleisch» aus. | In der Tabelle sollte nur noch das «Rind» erscheinen. | y |
| Klicken Sie auf «Übersicht». Wählen Sie im Dropdown «Alle» aus. | In der Tabelle sollte das «Rind» und der «Backfisch» erscheinen. | y |
| Klicken Sie auf den Knopf «Verbrauchen». Wählen Sie «Backfisch vom …» aus. Geben Sie anschliessend im «Anzahl» Feld «2» ein. Klicken Sie auf «Verbrauchen». | Der Artikel sollte aus der Tabelle verschwinden. | Y |

# Auswerten

## Erreichen der Ziele

Nachstehend wird evaluiert, welche der Must-Haves und Nice-to-Haves ich erreicht habe.

### Must-Have

Das Hinzufügen wie auch das Verbrauchen/Löschen von Produkten zur Datenbank wurde erreicht. Benutzerdefiniert Kategorien zu erstellen ist ebenfalls möglich, wohingegen zwei Kategorien und Artikel bereits in die Datenbank hinzugefügt wurden. Das Benachrichtigen des Users, falls Artikel eventuell Artikel abgelaufen sein könnten, funktioniert ebenfalls einwandfrei.

### Nice-to-Have

Das einzige Nice-to-Have welches ich mir vorgenommen habe, habe ich leider nicht implementieren können/wollen. An sich würde ein Barcode Scanner funktionieren, jedoch habe ich den zeitlichen Aufwand vom Rest des Projektes anders eingeschätzt. Ebenso finde ich, dass das Feature zu umständlich für den Endnutzer gewesen wäre, weil er dann jedes Mal ein paar Sekunden warten müsste, bis die Kamera läuft und der Barcode gescannt wurde.

## Known Bugs/Problems

* Wenn keine Daten in TArtikel vorhanden sind, wird bei Starten der App ein SQL error erscheinen
* Kategorien können nicht gelöscht werden
* Kategorien können nicht beim Bearbeiten eines Artikels geändert werden.

## Probleme

Eine Semesterarbeit geht nicht ohne Problem von statten. Diese werden im Folgenden Unterkapitel erläutert.

### Zeit

Das grösste Problem, welches ich hatte, war dass ich die Zeit ein wenig unterschätzt habe. Jedoch bin ich mit dem Zeitbudget, welches ich noch übrighatte, zufrieden, da ich dennoch ziemlich gut fertig geworden bin.

## Cordova

Das grösste Problem, welchem ich gegenüber stand, war, als ich das erste Mal versucht habe, eine .APK Datei zu erstellen. Es stellte sich heraus, dass ich einerseits ein notwendiges Programm noch nicht installiert hatte. Ebenso verlangt Cordova eine ganz bestimmte Version von einem Programm, welche ich nur durch langes Suchen in den erweiterten Einstellungen des Programms gefunden und anschliessend installiert hatte. Nachdem dies gelöst war, lief die ganze Entwicklung mit Cordova reibungslos ab.

## Fazit

Diese Semesterarbeit war bisher mein umfangreichstes Projekt, jedoch kann ich mit Stolz sagen, dass ich zufrieden bin. Ich habe das erreicht, was ich machen wollte, und finde, dass meine App auch nach der Abgabe des Projektes einen Sinn und einen Nutzen hat. Leider habe ich nicht alle Funktionen implementieren können, jedoch bedaure ich das nicht allzu sehr, da diese Funktionen die Nutzerfreundlichkeit eventuell zerstört hätten. Weiterführend fand ich es grossartig, mich noch einmal mit Cordova auseinanderzusetzen und diese erweitert mit SQLite verwendet zu haben. Im Grossen und Ganzen kann ich sagen, dass dieses Projekt bisher am interessantesten, nervenaufreibendsten und befriedigendsten von all den Projekten war, welche ich bisher gemacht hatte.

# Anleitungen

## Installation

1. Freezing\_Inventory.apk von der GitHub release Page herunterladen
2. Im Dateimanager des Handys auf die .apk Datei klicken und per Paketinstallation installieren
3. Warnung Ignorieren (Wird immer bei .apk Dateien angezeigt)

## Benutzeranleitung

**Übersicht**: Auf dieser Seite kann man all seine Artikel sehen, welche man in der Datenbank hat. Im Dropdown Menu kann man auswählen nach was für einer Kategorie gefiltert werden soll.

**Verbrauch:** Auf dieser Seite kann man in der Dropdownliste einen Artikel auswählen, welchen man verbrauchen möchte. Im Feld «Anzahl» kann man dann die Anzahl angeben, welche man verbrauchen will und dann auf «Verbrauchen» klicken. Folglich reduziert es den Artikel um besagte Anzahl und wenn die Zahl 0 und kleiner ist, wird der Artikel gelöscht.

**Verwalten**:

**Hinzufügen**: Auf dieser Seite kann man neue Artikel wie auch neue Kategorien hinzufügen. Wenn man einen neuen Artikel hinzufügen möchte, wählt man eine Kategorie aus der Dropdownliste aus und gibt entsprechende Eckdaten wir Name, Anzahl und Ablaufdatum an und fügt diese durch den Knopf «Hinzufügen» hinzu. Wenn man jedoch eine neue Kategorie hinzufügen möchte, wählt man in dem Dropdownmenu «Neue Kategorie» aus. Folglich schreibt man in das Feld «Kategoriename» den Namen der Kategorie hinein, welche man hinzufügen möchte und drückt anschliessend auf den Knopf «Hinzufügen».

**Bearbeiten**: Auf dieser Seite kann man die Eckdaten eines Artikels überarbeiten. Man wählt aus der Dropdownliste den gewünschten Artikel und die Eckdaten werden folglich in die gleichnamigen Eingabefelder geschrieben. Anschliessend kann man in den Eingabefeldern die Eckdaten verändern und wenn man den Prozess abschliessen möchte, drückt man auf den Knopf «Speichern».

**Löschen**: Diese Seite funktioniert genau gleich wie die Verbrauch Seite. Man wählt einen Artikel aus der Dropdownliste aus und gibt man wie viele man davon verbrauchen will und drückt abschliessend auf «Löschen». Die Artikel werden nur gelöscht, wenn die Verbrauchsanzahl grösser ist als die Mengenanzahl des Artikels.

# Abbildungsverzeichnis

Abbildung 1 Android Logo

Abbildung 2 IOS Logo

Abbildung 3 Native

Abbildung 4 Webapp

Abbildung 5 Hybrid

Abbildung 6 Firebase Logo

Abbildung 7 SQLite Logo

Abbildung 8 Cordova Logo

Abbildung 9 Flutter Logo

Abbildung 10 Mock-Up

Abbildung 11 Storyboard

Abbildung 12 ERD

Abbildung 13 GitHub Logo