Zadanie domowe 2

Systemy i aplikacje bez granic – wersja 1.2 z 03.05.2020 13:43:00

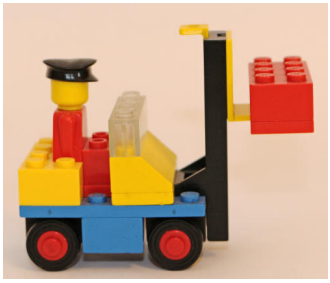
Aplikacja BrickList

Opis problemu

W wielkim pudle z klockami Lego znajduje się wiele wymieszanych ze sobą zestawów. Prawdopodobnie też nie są już one kompletne, tzn. może brakować niektórych klocków. Za cel stawiamy sobie odbudowanie/skompletowanie wybranego zestawu. Pomoże nam w tym Internet, a później nasza aplikacja.
Pierwszy krok do odbudowania zestawu to określenie jakie klocki się na niego składają. W przypadku nowszych zestawów (z XXI wieku), nawet jeżeli straciliśmy oryginalną instrukcję, instrukcje znajdziemy w sieci na stronie producenta.
Załóżmy, że chcemy odbudować zestaw nr 70403 – Smocza Góra.

Na końcu instrukcji znajdziemy listę klocków, które składają się na ten zestaw:

Możemy sobie taką listę wydrukować i odznaczać, które klocki udało się już nam znaleźć, ale w praktyce taka forma nie jest wygodna.
Sytuacja wygląda gorzej, jak chcemy odtworzyć stary zestaw. Tutaj instrukcji z listą klocków nie znajdziemy. Z pomocą przychodzi nam Internet i różne serwisy, które posiadają bazy danych zestawów i ich zawartości (Inventory). Jednym z takich serwisów jest BrickLink.
Załóżmy, że chcemy odbudować zestaw nr 615 – Wózek widłowy z roku 1975.



Na BrickLink możemy sobie wygenerować do wydruku listę klocków, której fragment widać poniżej.

23.04.2018, 11:07

Catalog: Sets: Legoland: Construction: 615-2: Inv

Fork Lift with Driver

*

This Set Consists of the following Items:

Chk	Image	Qty	Item No	Description	MID
				Regular Items:	
				Parts:	
☐	 *	2	3137c01assy1 (Inv)	**Black Brick, Modified 2 x 2 with Wheels Red for Single Tire, with Black Tires 14mm D. x 4mm Smooth Small Single (3137c01 / 3139)** Catalog: Parts: Wheel & Tire Assembly	
☐	 * !	1	3430c02 (Inv)	**Black Vehicle, Forklift 2 x 2 Plate and Yellow Fork (Complete Assembly)** Catalog: Parts: Vehicle	
☐	 *	1	3001old	**Blue Brick 2 x 4 without Cross Supports** Catalog: Parts: Brick	
☐	 * !	1	3032	**Blue Plate 4 x 6** Catalog: Parts: Plate	
☐	 *	1	3001old	**Red Brick 2 x 4 without Cross Supports** Catalog: Parts: Brick	
☐	 * !	2	3023	**Red Plate 1 x 2** Catalog: Parts: Plate	
☐	 * !	1	3010	**Trans-Clear Brick 1 x 4** Catalog: Parts: Brick	
☐		1	3001old	**Yellow Brick 2 x 4 without Cross Supports** Catalog: Parts: Brick	

Ale znowu, papier nie jest najwygodniejszą formą zaznaczania, jakich klocków i ile nam się udało odnaleźć.
Docelowo oczywiście będziemy chcieli kupić brakujące klocki w jednym z serwisów oferujących taką możliwość i ma nam w tym pomóc aplikacja.

Aplikacja BrickList

Aplikacja ma być odpowiednikiem papierowej listy klocków, ale oczywiście ma umożliwiać znacznie więcej.
Po pierwsze – w aplikacji mamy możliwość pracy nad wieloma projektami na raz, czyli tak, jakbyśmy takich kartek ze spisem mieli dowolną ilość.
Po drugie – w aplikacji mamy mieć możliwość zaznaczania (pole tekstowe Plain Text, może z przyciskami +/-) informacji o tym, ile klocków danego typu udało się nam już odnaleźć, ewentualnie, ilu nam nadal brakuje. Przy tym dobrze by było, gdyby klocki można grupować ze względu na typ (np. klocek 2x4 – jego ItemID) lub kolor.
Po trzecie – gdy się okaże, że nie udało się nam znaleźć wszystkich klocków aplikacja ma przygotować plik XML z listą brakujących klocków i umożliwić jego przesłanie poza urządzenie (zapis na kartę SD, wysłanie mailem a może zapisanie na dysku sieciowym – Storage Access).
W przyszłości mamy mieć możliwość powrotu do listy i zaznaczenia informacji np. o klockach, które udało się nam dokupić.

Scenariusz użycia

Załóżmy, że chcemy odtworzyć nasz zestaw 615. Skąd weźmiemy listę klocków? Po zalogowaniu się w serwisie BrickLink, w sekcji Download można ściągać w postaci CSV lub XML listę klocków z zestawu (Inventory). My rozwiążemy ten problem na razie w ten sposób, że tworząc nowy projekt, będziemy wprowadzać URL do pliku XML, przy czym możemy założyć, że główna część URL'a jest zapisywana w ustawieniach aplikacji, a nazwa samego pliku XML jest już wprowadzana przez użytkownika, czyli będziemy budować URL z dwóch stringów. Na potrzeby zadania plik z inventory zestawu 615 jest dostępny pod adresem: http://fcds.cs.put.poznan.pl/MyWeb/BL/615.xml. Aplikacja pamiętałaby adres „http://fcds.cs.put.poznan.pl/MyWeb/BL/" , a użytkownik wprowadzałby tylko „615". Rozszerzenie może się dodać samo. Pod tym samym adresem znajdują się też pliki dla zestawów: 70403 – Smocza Góra, 10179 – Sokół Millenium UCS, 361 – Kawiarenka, 10258 – Londyński Autobus, 384 – Londyński Autobus i 555 – Szpital.

Po załadowaniu pliku, mamy nowy projekt, nad którym możemy pracować w aplikacji, który powinien się pojawić na liście projektów. Po otwarciu projektu, mamy widok listy podobny do tej listy z BrickLink, tyle, że w każdym wierszu mamy możliwość określenia, ile klocków udało się nam znaleźć (przydatne przyciski +/-). Jeżeli znaleźliśmy wszystkie klocki danego typu, to taki wiersz podświetlałby się w innym kolorze, a dodatkowo mógłby wędrować na koniec listy tak, aby na początku były tylko wpisy z brakującymi klockami (to wędrowanie jest opcjonalne).

Gdy użytkownik dojdzie do wniosku, że więcej klocków już nie uda mu się znaleźć, może wygenerować plik XML z listą brakujących klocków i np. zapisać go na karcie SD, dysku sieciowym, lub ewentualnie wysłać mailem.

Format pliku z inventory

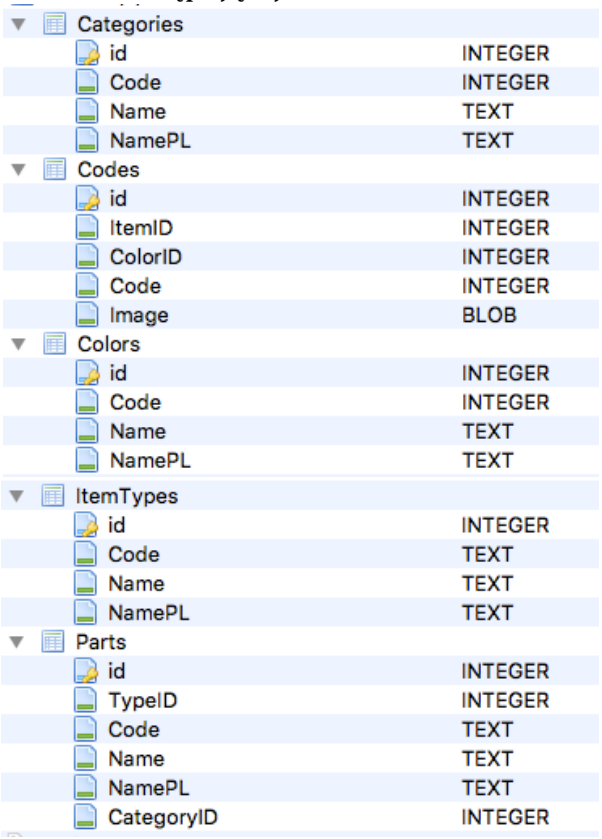
```
<?xml version="1.0" encoding="UTF-8"?>
<INVENTORY>
   <ITEM>
      <ITEMTYPE>M</ITEMTYPE>
      <ITEMID>old012</ITEMID>
      <QTY>1</QTY>
      <COLOR>0</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3430c02</ITEMID>
      <QTY>1</QTY>
      <COLOR>11</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3001old</ITEMID>
      <QTY>1</QTY>
      <COLOR>7</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3032</ITEMID>
      <QTY>1</QTY>
      <COLOR>7</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3010</ITEMID>
      <QTY>1</QTY>
      <COLOR>12</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3001old</ITEMID>
      <QTY>1</QTY>
```

```
      <COLOR>5</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3023</ITEMID>
      <QTY>2</QTY>
      <COLOR>5</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3001old</ITEMID>
      <QTY>1</QTY>
      <COLOR>3</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3023</ITEMID>
      <QTY>2</QTY>
      <COLOR>3</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3037</ITEMID>
      <QTY>1</QTY>
      <COLOR>3</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
   <ITEM>
      <ITEMTYPE>P</ITEMTYPE>
      <ITEMID>3137c01assy1</ITEMID>
      <QTY>2</QTY>
      <COLOR>11</COLOR>
      <EXTRA>N</EXTRA>
      <ALTERNATE>N</ALTERNATE>
      <MATCHID>0</MATCHID>
      <COUNTERPART>N</COUNTERPART>
   </ITEM>
</INVENTORY>
```

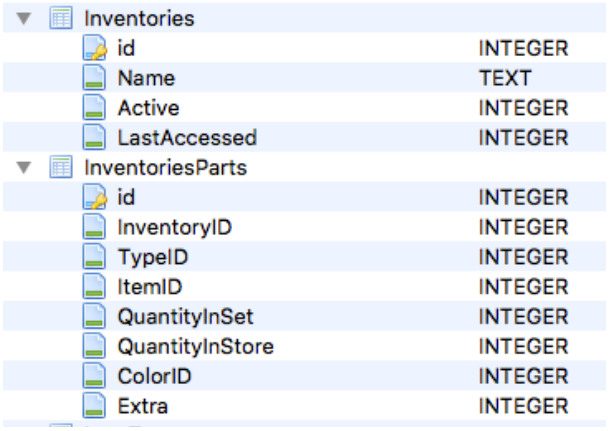
Jak widać w tym pliku za wiele informacji nie mamy. Konkretny klocek jest identyfikowany przez ITEMID (np. 3001 oznacza klocek 2x4 ale tej informacji w pliku nie ma) oraz COLOR. Dodając elementy do listy, możemy pominąć te, które mają znacznik ALTERNATE o innej wartości niż N. Pomijamy też informacje z MATCHID i COUNTERPART. Skąd zdobyć resztę informacji? BrickLink udostępnia je w osobnych plikach. Natomiast dla Państwa wygody zebrałem je wszystkie w bazie danych SQLite, która ma też służyć do przechowywania projektów i być podstawą działania aplikacji.

Baza danych

Plik bazy ze wszystkimi potrzebnymi do pracy informacjami znajduje się pod adresem: http://fcds.cs.put.poznan.pl/MyWeb/BL/bricklist.zip.
Baza danych zawiera 7 tabel o następującej strukturze:



Powyższe tabele są już wstępnie wypełnione danymi.



W tych tabelach będziecie Państwo przechowywać dane dotyczące poszczególnych projektów.
Polecam korzystanie z narzędzia SQLite Database Browser: http://sqlitebrowser.org/

Opis tabel

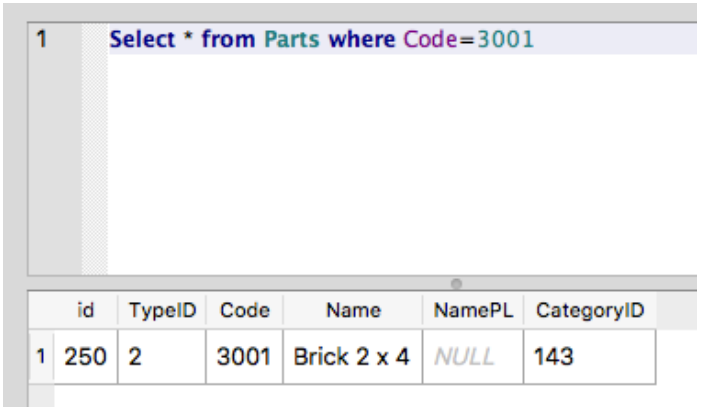
Tabela **Categories** zawiera kategorie, do których mogą należeć różne elementy. W naszej aplikacji praktycznie nie będziemy raczej z niej korzystać.
Tabela **Colors** zawiera nazwy kolorów, a także kod koloru, który występuje w pliku XML z inventory.
Tabela **ItemTypes** zawiera nazwy typów elementów, np. klocek, instrukcja, gadżety, książka, itp.
Tabela **Parts** zawiera najważniejsze informacje dotyczące klocków, w szczególności kod klocka (Code, taki jak w pliku XML) i nazwę klocka.
Tabel **Codes** łączy w sobie informacje o kolorze klocka i rodzaju klocka, co pozwala pobrać jego unikalny identyfikator DesignID (pole Code), który unikalnie identyfikuje dany klocek. Niestety starsze klocki nie posiadają swoich DesignID. W tej tabeli jest też miejsce na zdjęcie miniaturki klocka – na razie puste. Zakładam, że miniaturki będą ściągane do bazy tylko wtedy, gdy będą potrzebne. Bez tego, rozmiar pliku z bazą zawierająca podstawowe miniaturki pobrane tylko ze strony Lego ma około 300MB.

Tabela **Inventories** ma zawierać poszczególne projekty użytkownika, identyfikowane przez nazwę z opcją ich dezaktywacji – nieaktywne nie będą już widoczne. Pole LastAccessed ma umożliwić sortowanie projektów w widoku po dacie, kiedy ostatnio były otwarte.
Tabela **InventoriesParts** zawierać ma informacje o poszczególnych klockach wchodzących w skład projektu. InventoryID wskazuje do którego Inventory należy ten wiersz. ItemID i ColorID wskazują klocek. QuantityInSet oznacza ilość klocków tego typu w zestawie, a QuantityInStore oznacza, ile klocków udało się znaleźć i początkowo powinno być równe 0. Extra odpowiada polu EXTRA z pliku. TypeID wskazuje na typ elementu w tabeli ItemTypes. Na bazie informacji ItemID i ColorID można wyszukać informacje w tabeli Codes, pozwalające pobrać obrazek klocka.

W kilku tabelach jest pole NamePL, które jest na razie przeważnie puste i potencjalnie w nim będą tłumaczenia nazw na polski. Aplikacja mogłaby sprawdzać, czy istnieje polska nazwa i wtedy ją wyświetlać, a w przeciwnym razie wyświetlać angielską.

Przykład

Podstawowy klocek typu 2x4 ma Code o wartości 3001. Ale ten sam kod mają wszystkie takie klocki w każdym dostępnym kolorze. Aby uzyskać DesignID klocka w konkretnym kolorze sprawdzimy w tabeli Codes wszystkie wpisy o odpowiednim ID klocka.



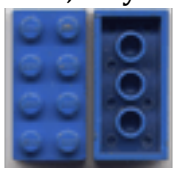
Nasz klocek ma identyfikator o wartości 250.
Przeszukujemy teraz tablicę Codes.

```
1  Select * from Codes where ItemID=250
```

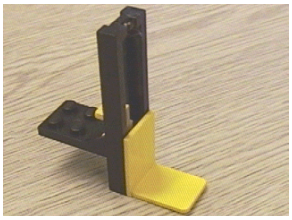
	id	ItemID	ColorID	Code	Image
1	7176	250	3	300126	NULL
2	7177	250	4	300123	NULL
3	7178	250	7	4216917	NULL
4	7179	250	8	4229356	NULL

Otrzymujemy w sumie 54 wiersze odpowiadające różnym kolorom tego klocka.
Skąd teraz wziąć obrazek? Np. ze strony Lego, gdzie można je znaleźć pod adresem https://www.lego.com/service/bricks/5/2/300126, gdzie końcówka odpowiada polu Code w tej tabeli.

Niestety nie zawsze znajdziemy taki obrazek. Na serwerach Lego niektórych brakuje, a już na pewno nie ma obrazków dla od dawna nieprodukowanych klocków. Co wtedy? Można skorzystać z zasobów BrickLink, gdzie obrazki znajdziemy pod adresem: http://img.bricklink.com/P/7/3001old.gif, gdzie 3001old, jest identyfikatorem starej wersji klocka 3001, a 7 oznacza kod koloru, w tym wypadku niebieski.



Gdy klocek nie ma wersji kolorystycznych to link może wyglądać tak: https://www.bricklink.com/PL/3430c02.jpg dla klocka o kodzie **3430c02**. Nie ma też wtedy wpisu w tabeli Codes.



Jak już się natrudzimy nad zdobyciem zdjęcia, to możemy je zapisać w tabeli Codes, w polu Image, żeby nie ściągać obrazków za każdym razem. Jeżeli wpisu nie było (patrz wyżej) to możemy go dodać.

Plik wyjściowy

Gdy użytkownik zechce już przygotować listę brakujących klocków powinna ona zostać zapisana w pliku XML w formacie tzw. WantedList. Opis formatu znajduje się tutaj: https://www.bricklink.com/help.asp?helpID=207.
W szczególności może on wyglądać tak:

```
<INVENTORY>
<ITEM>
<ITEMTYPE>P</ITEMTYPE>
<ITEMID>3622</ITEMID>
<COLOR>11</COLOR>
<QTYFILLED>4</QTYFILLED>
</ITEM>
<ITEM>
<ITEMTYPE>P</ITEMTYPE>
<ITEMID>3001</ITEMID>
<COLOR>5</COLOR>
<QTYFILLED>10</QTYFILLED>
<CONDITION>N</CONDITION>
</ITEM>
</INVENTORY>
```

Musimy wypełnić minimalnie ITEMTYPE, ITEMID, COLOR. Potrzebne jest też QTYFILLED oznaczające liczbę potrzebnych klocków. Opcjonalnie możemy dodać CONDITION, które oznacza, czy klocek ma być nowy (N), czy może być używany (U) – może to być parametr eksportu (tylko nowe, tylko używane, nieważne).

Jak zapisać XML?

```
fun writeXml() {

    val docBuilder: DocumentBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder()
    val doc: Document = docBuilder.newDocument()

    val rootElement: Element = doc.createElement( tagName: "person")

    rootElement.setAttribute( name: "person-id",  value: "1001")


    val lastName: Element = doc.createElement( tagName: "last-name")

    lastName.appendChild(doc.createTextNode( data: "Doe"))

    rootElement.appendChild(lastName)

    val firstName: Element = doc.createElement( tagName: "first-name")

    firstName.appendChild(doc.createTextNode( data: "John"))

    rootElement.appendChild(firstName)

    doc.appendChild(rootElement)

    val transformer: Transformer = TransformerFactory.newInstance().newTransformer()

    transformer.setOutputProperty(OutputKeys.INDENT,  value: "yes")
    transformer.setOutputProperty( name: "{http://xml.apache.org/xslt}indent-amount",  value: "2")

    val path=this.filesDir
    val outDir = File(path,  child: "Output")
    outDir.mkdir()

    val file=File(outDir,  child: "test.xml")

    transformer.transform(DOMSource(doc), StreamResult(file))
}
```

Interfejs użytkownika

Na interfejs powinny składać się następujące aktywności:

1. Ekran główny – lista projektów z możliwością dodania nowego projektu i zarchiwizowania istniejącego (domyślnie zarchiwizowane nie są widoczne).
2. Ekran zestawu – podstawowy ekran z listą klocków, możliwością wpisywania informacji o znalezionych klockach.
3. Ekran dodawania nowego projektu z adresu URL.
4. Ekran ustawień aplikacji, np. prefiks adresu URL.
5. Ekran eksportu pliku XML z brakującymi klockami (chyba, że rozwiążecie to jakoś inaczej).

Uwagi

Przy imporcie może, choć jest to mało prawdopodobne, okazać się, że danego klocka nie mamy w bazie – powinien się wtedy pojawić odpowiedni komunikat na ekranie (z ITEMID klocka i jego kolorem) i dalej sprawę ignorujemy.

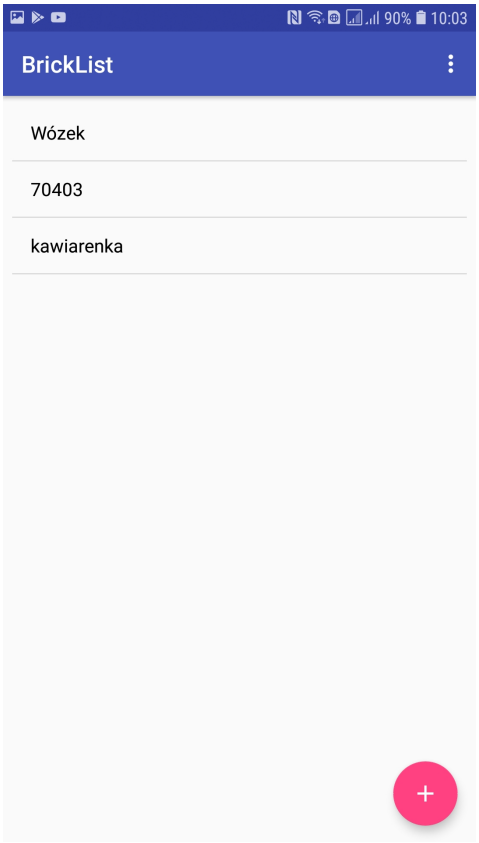
Gdy nie uda się nam znaleźć obrazka klocka, nie przejmujemy się tym.

Dopuszczam niewielkie zmiany w strukturze bazy, gdyby okazały się konieczne.

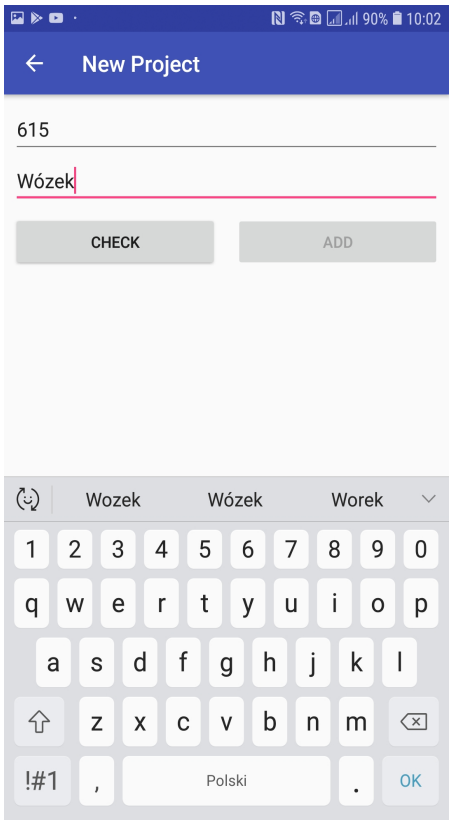
Do praktycznego działania potrzebny byłby mechanizm aktualizacji bazy, ale tym się na razie nie przejmujemy.

Przykładowe proponowane ekrany:

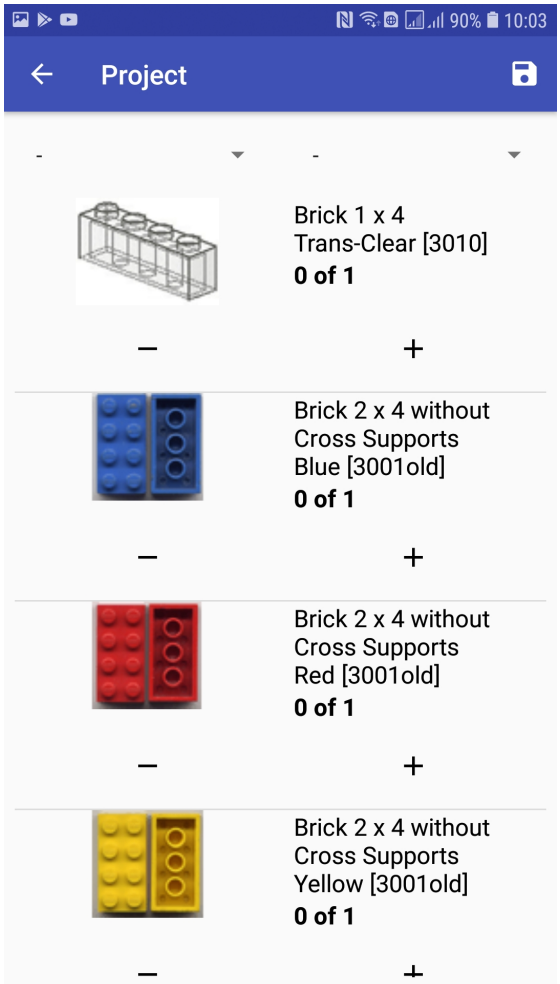
Ekran główny:



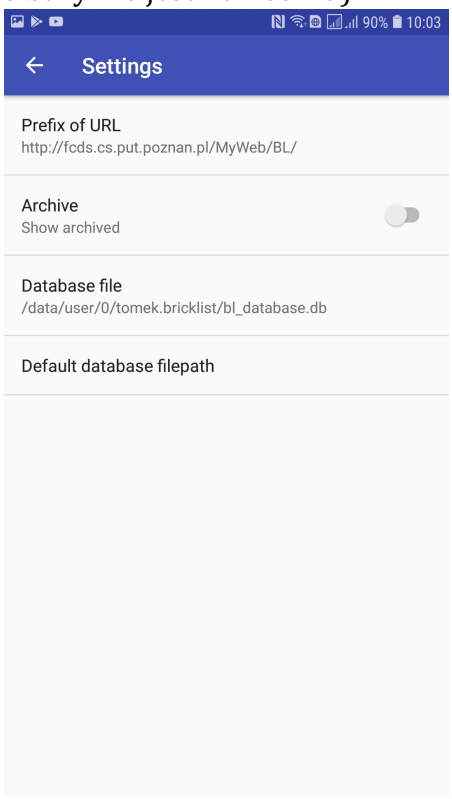
Dodawanie projektu:



Widok projektu:



Ustawienia (baza i ścieżka do bazy nie jest konieczna):



Termin oddania zadania: 25 maja 2020 r.

Jeżeli uczelnia po 24 maja zostanie otwarta, to zadania będzie należało oddać na zajęciach w tygodniu 25-29 maja.
Jeżeli nie, to proszę je odsyłać jak poprzednio mailem zatytułowanym [UBI]Zadanie domowe 2. Najlepiej link do projektu na dysku sieciowym lub ewentualnie link do repozytorium GIT. Proszę pamiętać przed wysłaniem o sprawdzeniu, jak działa aplikacja na „wyczyszczonym” symulatorze, żeby nie było niespodzianek. Zadania zacznę sprawdzać po 25 maja.