

Bazy danych 2 - Projekt

Bunch Of Tools

**Projekt systemu wspomagającego wypożyczalnię
sprzętu budowlanego z wykorzystaniem
relacyjnej bazy danych, REST API oraz aplikacji
mobilnej na system Android.**

Wykonanie:

Mateusz Wocka 218260

Tomasz Czart 218335

Damian Nowak 218197

Repozytorium projektu:

https://github.com/damiannowak01011111/Project_DataBase2_2016_2017

1. Spis treści

1.	Spis treści	2
2.	Wstęp	3
2.1.	Opis biznesowy systemu	3
2.2.	Struktura systemu	4
2.3.	Stos technologiczny	5
3.	Baza danych	7
4.	REST API	18
5.	Aplikacja mobilna	22
6.	Podsumowanie	29

2. Wstęp

Celem naszego projektu było zaprojektowanie oraz implementacja systemu wspomagającego pracę wypożyczalni sprzętu budowlanego. Robocza nazwa wypożyczalni to *Bunch Of Tools*.

2.1. Opis biznesowy systemu

Opis zasobów ludzkich.

Pracownik wypożyczalni sprzętu budowlanego może dodawać do katalogu nowe modele sprzętu. Może również dodawać i usuwać poszczególne egzemplarze, edytować ich stan i status. Pracownik jest również odpowiedzialny za wydanie sprzętu klientowi, co notuje w systemie, oraz jego zdanie przez klienta co także odnotowuje do systemu. Klient może rezerwować sprzęt, posiada unikalny numer oraz historie transakcji. Model sprzętu posiada swój numer, nazwę, opis i cenę za godzinę wypożyczenia. Dany egzemplarz sprzętu posiada informacje o przynależności do modelu, swój stan zużycia oraz status dostępności. Firma prowadzi spis wszystkich transakcji, nadając im unikalne numery. Każda transakcja ma numer klienta, numery egzemplarzy sprzętu, datę wypożyczenia i oddania oraz uiszczoną opłatę.

Przepisy, strategia firmy.

Firma nie pozwala jednemu klientowi na rezerwację więcej niż 10 sztuk sprzętu. Okres rezerwacji to 7 dni. Każdy klient jest zobowiązany wypełnić protokół zdawczo-odbiorczy. Pracownik ponosi odpowiedzialność za poprawność danych - odpowiada materialnie za niezgodność danych ze stanem fizycznym sprzętu w wypożyczalni. System dokonywania rezerwacji powinien być przyjazny dla klienta indywidualnego.

Dane techniczne

Klient może przeglądać sprzęt oraz dokonywać rezerwacji za pomocą aplikacji mobilnej. Zakłada się, że klientów, jednocześnie przeglądających asortyment za

pomocą aplikacji oraz dokonujących transakcji może być ponad 500. Firma wypożyczająca może oferować kilkaset typów produktów.

Lista wymagań funkcjonalnych

- Rejestracja klienta / pracownika do systemu.
- Logowanie klienta / pracownika do systemu.
- Możliwość przejścia do panelu administracyjnego dla użytkownika z uprawnieniami pracowniczymi.
- Przeglądanie asortymentu wypożyczalni z wyszczególnieniem podziału na kategorie sprzętu, model oraz poszczególne egzemplarze.
- Możliwość przeglądania składania rezerwacji przez użytkowników z uprawnieniami klienta.

Lista wymagań нефункциональных

- System powinien być łatwo rozszerzalny o nowe funkcjonalności
- Aplikacja powinna posiadać nowoczesny i przyjazny interfejs (preferowany Material Design)
- Aplikacja powinna być wysoce responsywna, a cały system wydajny. Maksymalny czas ładowania się danych przy sprzyjających okolicznościach powinien być poniżej jednej sekundy.

2.2. Struktura systemu

System wypożyczalni opiera się na trzech głównych filarach:

- Relacyjna baza danych postawiona na serwerze lokalnym przy wykorzystaniu oprogramowania *Microsoft SQL Server 2014*. Baza ma za zadanie przechowywanie danych takich jak: konta klientów, pracowników, dane o asortymencie z uwzględnieniem jego statusu, zdjęcia poszczególnych modeli sprzętu jak i historię transakcji dokonanych w sklepie. Baza danych ma również jak najbardziej odciążać REST API oraz aplikacje mobilną przy zarządzaniu danymi przez co zawiera zestaw triggerów automatyzujących zarządzanie danymi pracą.

- RESTful Web Services - REST API oparte na technologii Node.js. API jest uruchomiane na tym samym lokalnym urządzeniu co serwer bazodanowy. Jako jedyne ma bezpośredni dostęp do bazy danych, do łączenia z nią wykorzystuje SQL Server Authentication. Udostępnia ona szereg (ściśle kontrolowanych przez API) operacji na bazie danych. Komunikacja pomiędzy web service a aplikacją odbywa się za pomocą protokołu http.
- Aplikacja mobilna przeznaczona na system Android. Jest ona punktem styku systemu z jego użytkownikami. Służy do prezentacji danych zawartych w bazie danych, umożliwia ich modyfikowanie oraz spełnia wszystkie wcześniej wymienione wymagania funkcjonalne i nie funkcjonalne.

2.3. Stos technologiczny

Lista technologii / narzędzi użytych podczas realizacji projektu:

Baza danych:

- Rozszerzony język strukturalny T-SQL¹
- Narzędzia z rodziny Microsoft SQL Server 2014²
- Przeglądarka tekstowa lynx³ (dostępna na systemie Linux CentOS) oraz skrypty shell-owe do kolekcji danych.
- Generatory danych

API:

- Node.js⁴ (serwer API)
- Web framework Express⁵
- JSON Web Tokens⁶ (system autoryzacji pomiędzy API a aplikacją).
- JetBrains WebStorm (IDE)
- Protokół HTTP

¹ <https://msdn.microsoft.com/en-us/library/bb510741.aspx>

² <https://www.microsoft.com/pl-pl/sql-server/sql-server-editions>

³ <https://linux.die.net/man/1/lynx>

⁴ <https://nodejs.org/en/>

⁵ <http://expressjs.com/>

⁶ <https://jwt.io/>

- JavaScript

Aplikacja:

- Android SDK
- Java
- Android Studio (IDE)

Do zarządzania projektem:

- System kontroli wersji GIT⁷
- Publicznie dostępne repozytorium na GitHub⁸-ie (hostingowy serwis internetowy przeznaczony dla projektów programistycznych).
- Trello⁹ – serwis internetowy wspomagający pracę zwinną, umożliwiający dzielenie się notatkami, pomysłami i zadaniami w zespole.

⁷ <https://git-scm.com/>

⁸ https://github.com/damiannowak01011111/Project_DaTaBase2_2016_2017

⁹ <https://trello.com/>

3. Baza danych

W projekcie została stworzona i zaimplementowana relacyjna baza danych, wraz z wypełnieniem jej przykładowymi danymi oraz ustawieniem ograniczeń w postaci dostępnych słów, jak i triggerów odpowiadających za automatyczne ustawianie wartości jak i zezwolenia na dane operacje.

Baza danych została zaimplementowana w programie „SQL Server Management Studio”. Zawiera ona osiem tabeli takich jak: *Category*, *Customer*, *Employee*, *Item*, *Model*, *Reservation*, *Transactions* oraz *TransactionItem*. Tabela *Model* została wypełniona przy użyciu danych pobranych ze strony internetowej¹⁰ przy wykorzystaniu tekstowej przeglądarki *lynx* dostępnej na systemie Linux CentOS. Przy jej użyciu wraz z prostymi skryptami shell-owymi, zostały stworzone pliki tekstowe, które później zaimportowane zostały do tabeli. Natomiast tabela *Customer* została wypełniona przy użyciu generatora danych¹¹ dostępnych w sieci. Wszystkie inne zmiany dotyczące tabeli *Customer* były wprowadzane za pomocą generacji odpowiednich komend przy użyciu prostych programów napisanych w C++. Poniżej znajduje się przykładowy program, który generuje komendy SQL, dzięki którym losowo aktualizowane są świeżo dodane kolumny jakimi są *itemstatus* oraz *condition*.

```
int i;
srand(time(0));
int warunek = item + ilosc;
for (warunek; item <= warunek; item++) {

i = rand() % 10000;
cout << i;

if (i % 3 == 0) {
plik2 << "INSERT INTO [dbo].[Item] ([model_id] ,[itemstatus] ,[condition]) VALUES
(' " << model << ", 'Available' , 'Good') " << endl;
}
else if (i % 3 == 1) {
plik2 << "INSERT INTO [dbo].[Item] ([model_id] ,[itemstatus] ,[condition]) VALUES
(' " << model << ", 'Rent ' , 'Normal') " << endl;
}else
plik2 << "INSERT INTO [dbo].[Item] ([model_id] ,[itemstatus] ,[condition]) VALUES
(' " << model << ", 'Available ' , 'Bad') " << endl;
}

system("pause");
```

Listing 1 Zautomatyzowanie generacji komend w SQL

¹⁰ <http://www.homedepot.com/>

¹¹ <http://www.generatedata.com/>

Poniżej został także umieszczony zrzut ekranowy strony z której zostały wygenerowane dane dla tabeli *Customer*.

DATA SET ?

Order	Table Column	Data Type	Examples	Options	Help	Del
1	name	Names	Alex (any gender)	Name	?	
2	address	Street Address	No examples available.	No options available.	?	
3	email	Email	No examples available.	No options available.	?	
4	phone	Phone / Fax	Different formats	1-Xxx-Xxx-xxxx Xxx-xxxx	?	

Order Table Column Data Type Examples Options Help Del

Add 1 Row(s)

EXPORT TYPES ?

CSV Excel HTML JSON LDIF Programming Language SQL XML - hide data format options

Database table name: Customer

Database Type: SQL Server

Misc Options:
 ☐ Include CREATE TABLE query
 ☐ Include DROP TABLE query
 ☐ Enclose table and field names with backquotes

Statement Type: ☒ INSERT ☐ INSERT IGNORE ☐ UPDATE

INSERT batch size: 10

Primary Key: ☐ None ☒ Add default auto-increment column

Rysunek 1 Zrzut ekranowy generatora danych

Dane w kolumnach takich jak *itemstatus* oraz *condition* w tabeli *Item* są ograniczone pod względem wpisywanych do nich wartości. Aby wykonać takie ograniczenia należy użyć komendy SQL -owej podanej poniżej. Niestety w SQL Server Management Studio nie występuje komenda *enum* dlatego w projekcie zostało to zastąpione komendą CHECK.

```
ALTER TABLE dbo.Item ADD CONSTRAINT check_itemstatus
CHECK (itemstatus IN('Rent', 'Reserved', 'Available'))
```

Listing 2 Przykładowa komenda, sprawdzająca poprawność wprowadzonych danych

Używając programu SQL Server Management Studio, zostały stworzone już wcześniej wymienione tabele. A cała baza danych jest znormalizowana do trzeciej postaci normalnej. Poniżej znajdują się wszystkie stworzone tabele wraz z opisem, oraz przykład ustawienia auto inkrementującego się klucza głównego.

- Tabela *Category*

Posiada ona klucz główny *category_id* oraz kolumny *name* i *description*. Przechowuje ona dane dotyczące kategorii sprzętu, który będziemy chcieli wypożyczyć. Kolumny odpowiednio *name* i *description* odpowiadają nazwie kategorii oraz jej opisie.

Column Name	Data Type	Allow Nulls
* category_id	int	<input type="checkbox"/>
name	varchar(50)	<input type="checkbox"/>
description	varchar(50)	<input type="checkbox"/>

Rysunek 2 Tabela *Category*

Aby ustawić klucz główny, który sam się inkrementuje należy w ustawieniach w tym wypadku *category_id* zmienić wartość opcji *Identity Specification* na *yes*. Poniżej znajduje się zrzut ekranu ustawionego klucza głównego który automatycznie się inkrementuje.

* Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

Rysunek 3 Ustawienie auto inkrementacji dla klucza głównego

- Tabela *Customer*

Posiada ona klucz główny *customer_id*, który jest analogicznie ustawiony jak ten z poprzedniej tabeli. Kolumny *name*, *address*, *email*, *phone*, *password* oraz *username* zawierają dane użytkownika. Na tym poziomie aplikacji również hasło nie jest szyfrowane ponieważ tym zajmuje się REST API. Kolumna *reservation_counter* przechowuje informacje o tym ile dany użytkownik zarezerwował przedmiotów. Maksymalna liczba przedmiotów, które można zarezerwować wynosi 10.

Column Name	Data Type	Allow Nulls
* customer_id	int	<input type="checkbox"/>
reservation_counter	numeric(10, 0)	<input checked="" type="checkbox"/>
name	nvarchar(50)	<input type="checkbox"/>
address	nvarchar(50)	<input type="checkbox"/>
email	nvarchar(50)	<input type="checkbox"/>
phone	nvarchar(50)	<input type="checkbox"/>
password	nvarchar(MAX)	<input type="checkbox"/>
username	nvarchar(50)	<input type="checkbox"/>

Rysunek 4 Tabela *Customer*

- Tabela Employee

Bardzo podobna do tabeli Customer lecz nie przechowuje ani numeru telefonu ani adresu email, a co najważniejsze nie posiada licznika rezerwacji przedmiotów. Klucz główny ustawiony jest na kolumnę *employee_id* i tak jak w poprzednich tabelach jest on ustawiony na automatyczną inkrementację.

Column Name	Data Type	Allow Nulls
<i>employee_id</i>	int	⌏
name	varchar(50)	⌏
address	varchar(MAX)	☑
password	nvarchar(50)	⌏
username	varchar(50)	⌏

Rysunek 5 Tabela Employee

- Tabela Item

Tabela ta przechowuje klucz główny *item_id* oraz klucz obcy *model_id*, dzięki któremu możemy odwoływać się do tabeli Model. Tabela Item posiada również kolumny *itemstatus* oraz *condition*, które są ograniczone o dozwolone słowa, które mogą być wpisane w te kolumny. Dla *itemstatus* są to: *Rent*, *Reserved*, *Available* natomiast dla *condition*: *Good*, *Normal*, *Bad*. Poniżej znajduje się zrzut ekranowy tabeli.

Column Name	Data Type	Allow Nulls
<i>item_id</i>	int	⌏
<i>model_id</i>	int	☑
<i>itemstatus</i>	varchar(50)	⌏
<i>condition</i>	varchar(50)	⌏

Rysunek 6 Tabela Item

- Tabela Model

Tabela ta przechowuje klucz główny *model_id* jak i klucz obcy *category_id*, dzięki któremu możemy odwoływać się do tabeli Category. Tabela ta posiada także kolumny *name*, *description*, *price_per_hour*, które zawierają nazwę modelu, jego opis oraz cenę za godzinę. Kolumna *picture* natomiast przechowuje obraz w bazie danych.

Column Name	Data Type	Allow Nulls
<i>model_id</i>	int	⌏
name	varchar(50)	⌏
description	varchar(MAX)	⌏
price_per_hour	float	⌏
picture	image	☑
<i>category_id</i>	int	☑

Rysunek 7 Tabela Model

Aby wstawić do tabeli Model zdjęcie, została wykorzystana komenda w SQL, która znajduje się poniżej.

```
INSERT INTO Model (name, description, price_per_hour, picture)
SELECT 'Cable Installer',
'Install low voltage lighting or hidden pet fence quickly and easily
with the EZ-Cable Installer. This machine actually installs the wire
or
cable and backfills while digging the trench.',
90.0, *
FROM OPENROWSET (BULK N'C:\path\Cable-Installer.jpg', SINGLE_BLOB) IMAGE
```

Listing 3 Komendy SQL pozwalające na dodanie zdjęcia oraz innych danych dla Modelu

- Tabela Reservation

Tabela ta odpowiada za przechowywanie informacji o rezerwacji sprzętu, posiada klucz główny *reservation_id* oraz dwa klucze obce *customer_id* i *item_id* które pozwalają się połączyć odpowiednio z tabelami Customer oraz Item. Tabela Reservation posiada także kolumnę *date*, która przechowuje datę wykonania rejestracji. Data ta pobierana jest za pomocą funkcji *getdate()* ustawioną w programie SQL Server Management Studio.

Column Name	Data Type	Allow Nulls
reservation_id	int	<input type="checkbox"/>
customer_id	int	<input type="checkbox"/>
item_id	int	<input type="checkbox"/>
date	datetime	<input type="checkbox"/>

Rysunek 8 Tabela Reservation

- Tabela Transactions

Tabela odpowiadająca transakcje, posiada ona klucz główny *transaction_id* oraz dwa klucze obce *customer_id*, *employee_id*, które umożliwiają połączenie z tabelami Customer i Employee. Tabela Transactions przechowuje informacje o tym który klient u jakiego wypożyczającego wybrał sprzęt, posiada także kolumnę *date_start* i *date_end* informujące nas o rozpoczęciu i zakończeniu wypożyczenia, a także posiada kolumnę *price*, która przechowuje informację o należytej zapłacie za sprzęt.

Column Name	Data Type	Allow Nulls
transaction_id	int	<input type="checkbox"/>
customer_id	int	<input type="checkbox"/>
employee_id	int	<input type="checkbox"/>
date_start	datetime	<input checked="" type="checkbox"/>
date_end	datetime	<input checked="" type="checkbox"/>
price	money	<input checked="" type="checkbox"/>

Rysunek 9 Tabela Transactions

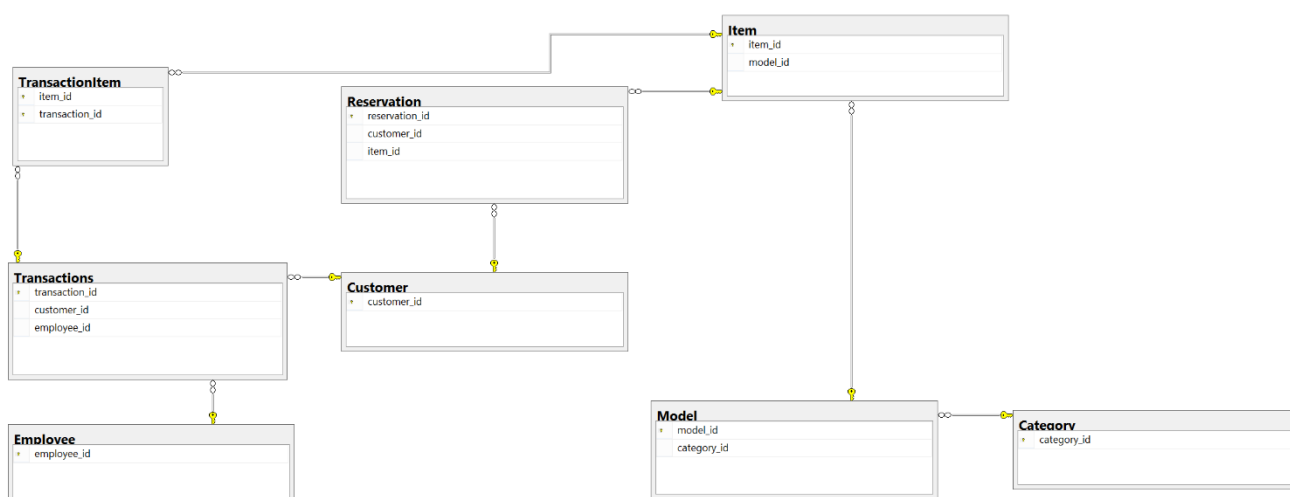
- Tabela TransactionItem

Tabela ta składa się tylko z klucza głównego, który jest złożeniem kluczy obcych, *item_id* oraz *transaction_id*. Tabela ta przechowuje informacje o wypożyczonych przedmiotach i numerach ich transakcji jest to najlepszy sposób na przechowywanie takich danych ponieważ pozbywamy się redundancji danych.

Column Name	Data Type	Allow Nulls
item_id	int	␣
transaction_id	int	␣

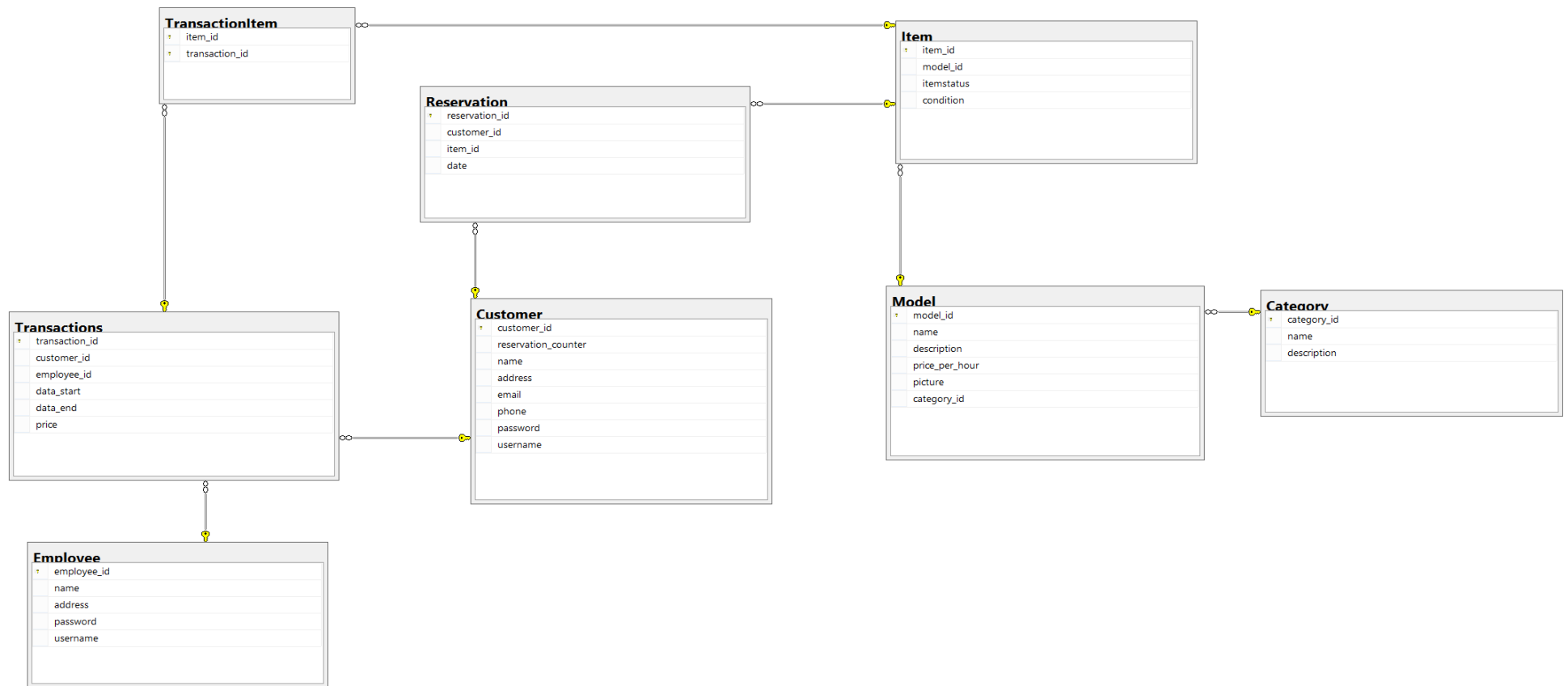
Rysunek 10 Tabela TransactionItem

Poniżej znajduje się diagram, pokazujący w graficzny sposób połączenie tabel przy użyciu kluczy obcych.



Rysunek 11 Diagram z kluczami głównymi i obcymi

Na kolejnej stronie znajduje się diagram bazy danych oraz graficzny sposób połączenia jej wykorzystując klucze obce.



Rysunek 12 Diagram bazy danych

W projekcie zostały także zaimplementowane triggery aby usprawnić powtarzalne zmiany w bazie danych.

- Trigger ReservationCounter

Dzięki niemu pole w tabeli Customer zwiększa się o jeden po każdym zarezerwowanym modelu, ma to na celu zwiększenie bezpieczeństwa przed zarezerwowaniem wszystkich dostępnych modeli. Trigger wyzwalany jest w momencie wywołania komendy Insert w tabeli Reservation. Sprawdza on czy klient nie przekroczył dozwolonej ilości zarezerwowanych przedmiotów, jeśli tak zwraca informację o zaistniałej sytuacji. Poniżej znajduje się kod triggera.

```
GO
CREATE TRIGGER ReservationCounter
ON Reservation
after Insert
AS
BEGIN
SET NOCOUNT ON
DECLARE @vname varchar(50);
SET @vname = (SELECT name FROM Customer WHERE Customer.customer_id in
(select customer_id from inserted))
IF (SELECT reservation_counter FROM Customer WHERE Customer.customer_id
in (select customer_id from inserted)) < 10
Update [dbo].[Customer] set reservation_counter = reservation_counter +1
where Customer.customer_id in (select customer_id from inserted)
ELSE
begin
PRINT @vname + ', You cannot order more Item.'
ROLLBACK TRANSACTION;
END
end
```

Listing 4 Trigger ReservationCounter

- Trigger ReservedStatus

Ten wyzwalacz ma na celu ustawienie wartości w *itemstatus* w tabeli Item na *Reserved* jeśli ta jest ustawiona na wartość *Available*. Trigger uruchamiany jest gdy w tabeli *Reservation* wprowadzana jest nowa dana.

```
GO
CREATE TRIGGER ReservedStatus
ON Reservation
after Insert
AS
BEGIN
SET NOCOUNT ON
IF (SELECT itemstatus FROM Item WHERE Item.item_id in (select item_id
from inserted)) = 'Available'
Update [dbo].[Item] set itemstatus = 'Reserved' where Item.item_id in
(select item_id from inserted)
ELSE
begin
ROLLBACK TRANSACTION;
END
end
```

Listing 5 Trigger ReservedStatus

- Trigger DeleteReservation

Wyzwalacz ten usuwa rezerwacje z tabeli *Reservation* oraz ustala wartość *Rent* w polu *itemstatus* w tabeli *Item*. Trigger jest wyzwalany w tabeli *TransactionItem* po wywołaniu komendy *Insert*.

```
GO
CREATE TRIGGER DeleteReservation
ON TransactionItem
after Insert
AS
BEGIN
SET NOCOUNT ON
IF (SELECT itemstatus FROM Item WHERE Item.item_id in (select item_id
from inserted)) = 'Reserved'
begin
Delete Reservation where Reservation.item_id in (select item_id from
inserted)
Update Item SET itemstatus = 'Rent' where Item.item_id in (select item_id
from inserted)
end
ELSE
IF (SELECT itemstatus FROM Item WHERE Item.item_id in (select item_id
from inserted)) = 'Available'
begin
Delete Reservation where Reservation.item_id in (select item_id from
inserted)
Update Item SET itemstatus = 'Rent' where Item.item_id in (select item_id
from inserted)
end
ELSE
IF (SELECT itemstatus FROM Item WHERE Item.item_id in (select item_id
from inserted)) = 'Rent'
begin
ROLLBACK TRANSACTION;
END
end
```

Listing 6 Trigger DeleteReservation

- Trigger AvaiaableStatus

Wyzwalacz ten ustawia wartość *Available* dla *itemstatus* w tabeli *Item*, po tym gdy pole *data_end* zostanie ustalone w tabeli *Transactions*. Trigger ten wywołuje się w momencie wykonania komendy *update* w tabeli *Transactions*.

```
GO
CREATE TRIGGER AvaiaableStatus
ON Transactions
after Update
AS
BEGIN
SET NOCOUNT ON
IF ((SELECT data_end FROM Transactions WHERE Transactions.transaction_id
in (SELECT transaction_id FROM TransactionItem WHERE
TransactionItem.transaction_id in (select transaction_id from
Transactions)))is not null)
begin
Update [dbo].[Item] set itemstatus = 'Available'
where Item.item_id in (SELECT item_id FROM TransactionItem WHERE
TransactionItem.transaction_id in (select transaction_id from inserted))
end
ELSE
begin
ROLLBACK TRANSACTION;
END
end
```

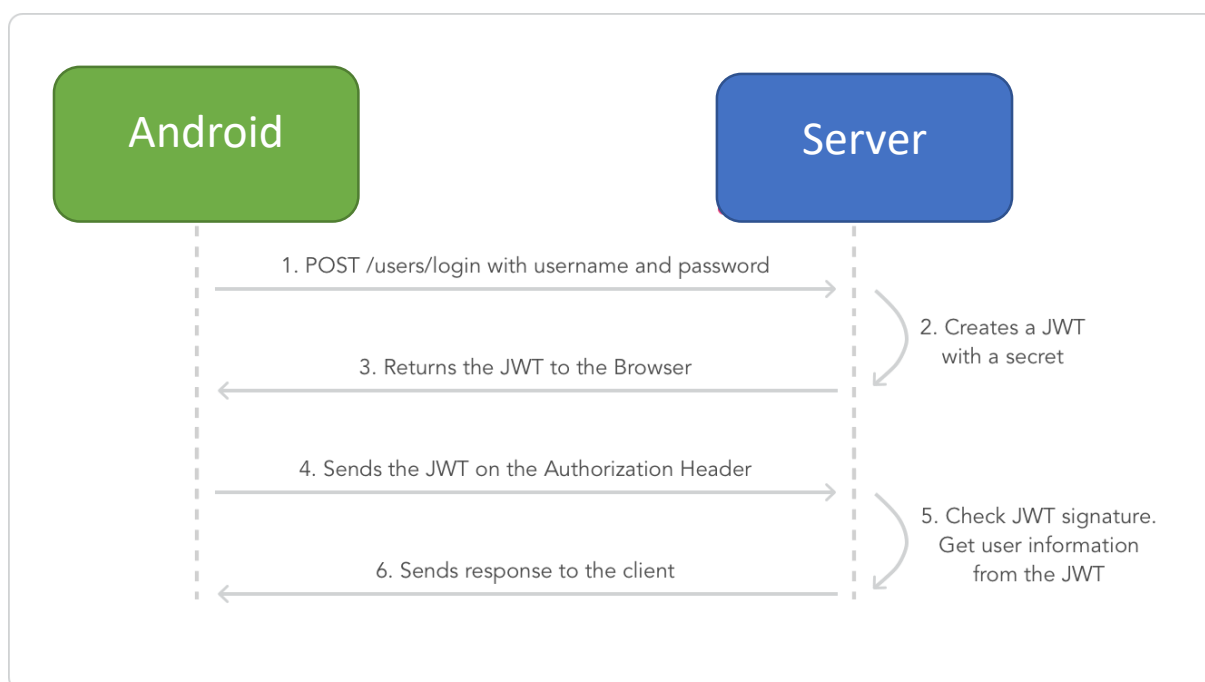
Listing 7 Trigger AvaiaableStatus

W każdym z powyższych triggerów wykorzystana została komenda *ROLLBACK TRANSACTION*, jest ona używana aby po negatywnym wyniku ze sprawdzenia warunku SQL nie wprowadził zmiany do tabel. Gdyby nie było tej komendy, to na przykład, w polu sprawdzającym stan rezerwacji w tabeli *Customer*, klient mógłby mieć wartość *reservation_count* powyżej liczby 10.

4. REST API

API zostało napisane w języku JavaScript i jest uruchamiane w środowisku Node.JS, do routingu używamy modułu express. Zastosowanie tego rozwiązania niesie ze sobą łatwość implementacji oraz szybką i niezawodną pracę pod dużym obciążeniem.

Uwierzytelnianie dostępu przez API odbywa się przy użyciu tokenu (JSON Web Token). Poniżej znajduje się rysunek prezentujący proces uwierzytelnienia z udziałem API oraz aplikacji.



Rysunek 133 Proces uwierzytelnienia przy wykorzystaniu JWT

Po zalogowaniu aplikacja mobilna otrzymuje odpowiedź od API z podpisanym loginem na pomocą tokenu. Token ten musi być wysyłany z każdym zapytaniem w celu weryfikacji uprawnień użytkownika. Token znajduje się w customowym nagłówku zapytania HTTP, "x-auth".

Token po rozkodowaniu ma taką strukturę:

```
{ "username": "Jan123", "employee": false }
```

Listing 8 Struktura tokenu

Hasła przechowywane w bazie danych są haszowane przez API za pomocą funkcji haszującej *BCRYPT*¹².

Korzyści płynące z użycia funkcji haszującej:

- Jest NIEODWRACALNA, nie ma ŻADNEJ możliwości odczytu hasła które zostało zahaszowane. Weryfikacja prowadzona jest na podstawie porównania zahaszowanych ciągów.
- Jest ona powolna, co utrudnia przeprowadzenia ataku hakerskiego.

Przykładowe metody udostępniane przez API:

Dane zwraca przez poniższe metody są przesyłane do aplikacji w formacie JSON¹³.

1. Rejestracja

Odbywa się po wysłaniu zapytania HTTP typu POST do API pod adres "127.0.0.1:3000/post/register". Do rejestracji wymagane są następujące elementy w sekcji body zapytania HTTP:

- employee true (employee) lub false (customer)
Pozostałe elementy nagłówka zależą od tego pola.

Jeżeli true rejestrujemy Employee:

- username
- name
- address
- password

Jeżeli false rejestrujemy Customera:

- username
- name
- address
- password
- phone
- email

2. Logowanie

Logowanie odbywa się po wysłaniu zapytania HTTP typu POST do API pod adres "127.0.0.1:3000/post/login". Do logowania wymagane są następujące elementy w sekcji body zapytania HTTP:

¹² <https://en.wikipedia.org/wiki/Bcrypt>

¹³ <http://www.json.org/json-pl.html>

- employee true (employee) lub false (customer)
- username
- password

Po poprawnym zalogowaniu serwer wysyła w odpowiedź HTTP o kodzie 200 która w sekcji body zawiera wygenerowany token. Token ten będzie musiał być wysyłany przy każdym kolejnym zapytaniu w celu autoryzacji użytkownika.

3. Dodawanie rezerwacji.

'/post/reservation' służy do tworzenia nowej rezerwacji. Rezerwację może stworzyć tylko użytkownik posiadający prawa Customera oraz którego licznik rezerwacji nie przekroczył limitu. W nagłówku zapytania x-auth należy przesłać token, a w body:

- customer_id
- item_id
- date (musi być w formacie 'YYYY-mm-dd HH:ii:ss')

4. Tworzenie transakcji

'/post/makeTransaction' służy do tworzenia nowej transakcji. Transakcję może stworzyć tylko użytkownik posiadający prawa Employee. W nagłówku zapytania x-auth należy przesłać token, a w body:

- customer_id
- item_id
- data_start (musi być w formacie 'YYYY-mm-dd HH:ii:ss')

5. Kończenie transakcji

'/post/endTransaction' służy do rozliczania transakcji. Transakcję może rozliczyć tylko użytkownik posiadający prawa Employee. W nagłówku zapytania x-auth należy przesłać token, a w body:

- transaction_id
- data_end (musi być w formacie 'YYYY-mm-dd HH:ii:ss')
- price

6. Pobranie listy modeli z zdjęciami

'/get/models' - lista modeli ze zdjęciami. Pobierane są następujące pola:

- model_id
- name
- description
- price_per_hour

- picture

7. Pobranie zdjęcia danego modelu

`'/get/imgOfModel/model_id'` - pobieranie obrazka o wskazanym `model_id`.

Aby pobrać obrazek dla wybranego modelu należy w adresie URL podać jego `model_id`. Przykładowo, pobranie obrazka dla modelu o `model_id=6` adres URL będzie następujący: `"/get/imgOfModel/6"`.

8. Pobieranie listy modeli z wybranej kategorii bez zdjęć

`'/get/modelsByCategory/category_id'` - pobieranie wszystkich modeli o wskazanym `category_id` bez zdjęć. Zapytanie zwraca następujące pola tablicy

- model_id
- name
- description
- price_per_hour

Przykład: dla `category_id=5` adres URL będzie następujący: `"/get/modelsByCategory/5"`.

9. Pobieranie listy itemów danego modelu

`'/get/itemsByModel/model_id'` - pobieranie wszystkich itemów o wskazanym `model_id`. Zapytanie zwraca następujące pola tablicy Items:

- model_id
- item_id
- itemstatus
- condition

Przykład, dla `model_id=12` adres URL będzie następujący: `"/get/itemsByModel/12"`.

5. Aplikacja mobilna

Aplikacja została napisana na system Android. Spełnia ona wszystkie wcześniej wymienione wymagania funkcjonalne i nie funkcjonalne. Do poprawnego funkcjonowania wymaga połączenia z REST API. Należy pamiętać o aktualizowaniu adresu do serwera w klasie *ApiConnectionFragment.java*:

```
public class ApiConnectionFragment extends Fragment {  
  
    public static final String URL_API = "http://192.168.1.169:3000";  
    .  
    .  
    .  
}
```

Listing 9 Zmienna URL_API z klasy ApiConnectionFragment zawierająca adres do serwera API

W poniższym listingu znajduje się funkcja końcowa dokonująca bezpośredniego połączenia z REST API. Należy zauważyć, że choć dana funkcja wykonuje połączenie, jest on ułamkiem całego procesu pobierania danych z API, nie widzimy tu parsowania danych z JSON-ów na klasy entity, obsługi błędów, sprzężenia zwrotnego oraz podziału wątków aby zachować responsywny UI. Łączna ilość kodu jest zbyt duża aby załączyć ją do sprawozdania dlatego znajduje się na ogólnodostępnym repozytorium.

```
private Result performTask(String requestMethod, URL url, String token,  
String postData) throws IOException {  
    InputStream stream = null;  
    HttpURLConnection connection = null;  
    Result result = new Result();  
    try {  
        connection = (HttpURLConnection) url.openConnection();  
        connection.setReadTimeout(10000);  
        connection.setConnectTimeout(10000);  
        connection.setRequestMethod(requestMethod);  
        connection.setDoInput(true);  
  
        connection.setRequestProperty("x-auth", token);  
  
        if (requestMethod.equals(REQUEST_METHOD_POST) &&  
!postData.equals("")) {  
            connection.setDoOutput(true);  
            OutputStream os = connection.getOutputStream();  
            BufferedWriter writer = new BufferedWriter(  
                new OutputStreamWriter(os, "UTF-8"));  
            writer.write(postData);  
  
            writer.flush();  
            writer.close();  
        }  
    }  
}
```

```

        // Open communications link (network traffic occurs here).
        connection.connect();
        publishProgress(ApiTaskCallback.Progress.CONNECT_SUCCESS);
        int responseCode = connection.getResponseCode();
        if (responseCode != HttpURLConnection.HTTP_OK && responseCode !=
201) {
            throw new IOException("HTTP error code: " + responseCode);
        }
        // Retrieve the response body as an InputStream.
        stream = connection.getInputStream();

        publishProgress(ApiTaskCallback.Progress.GET_INPUT_STREAM_SUCCESS, 0);
        if (stream != null) {
            result.mResultValue = readStream(stream, 100000);
        }
    } finally {
        // Close Stream and disconnect HTTPS connection.
        if (stream != null) {
            stream.close();
        }
        if (connection != null) {
            connection.disconnect();
        }
    }
    return result;
}

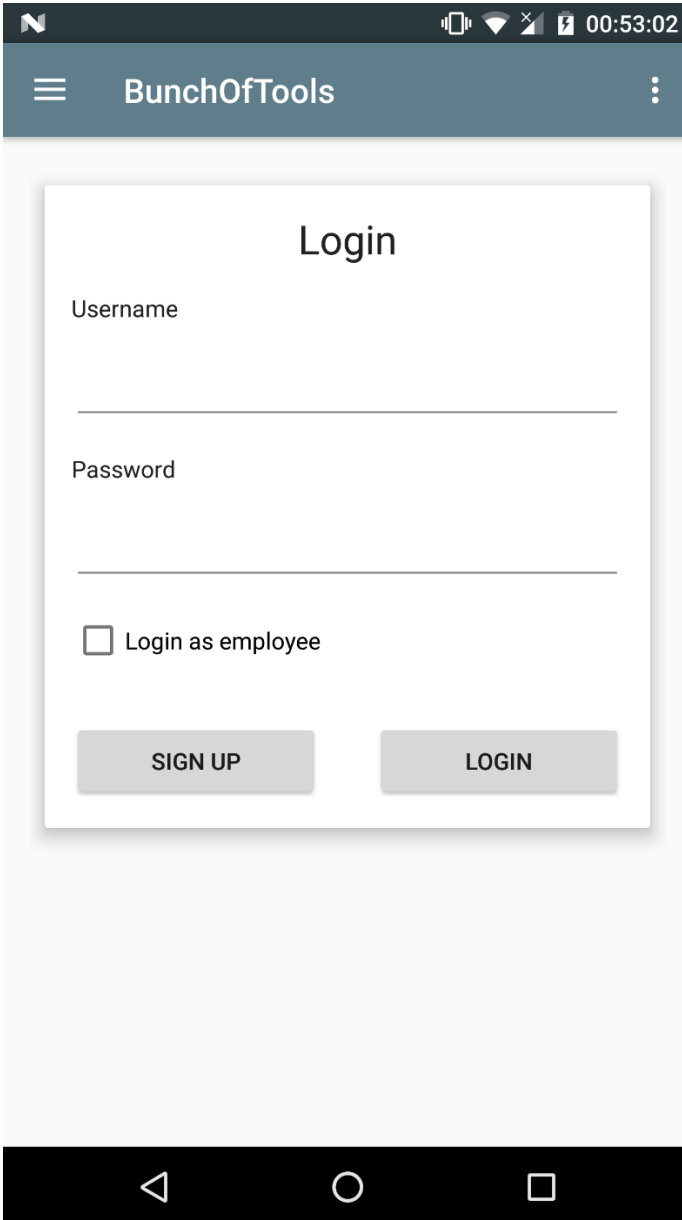
```

Listing 10 Funkcja realizująca połączenie z API

Przykład korzystania z aplikacji

1. Logowanie

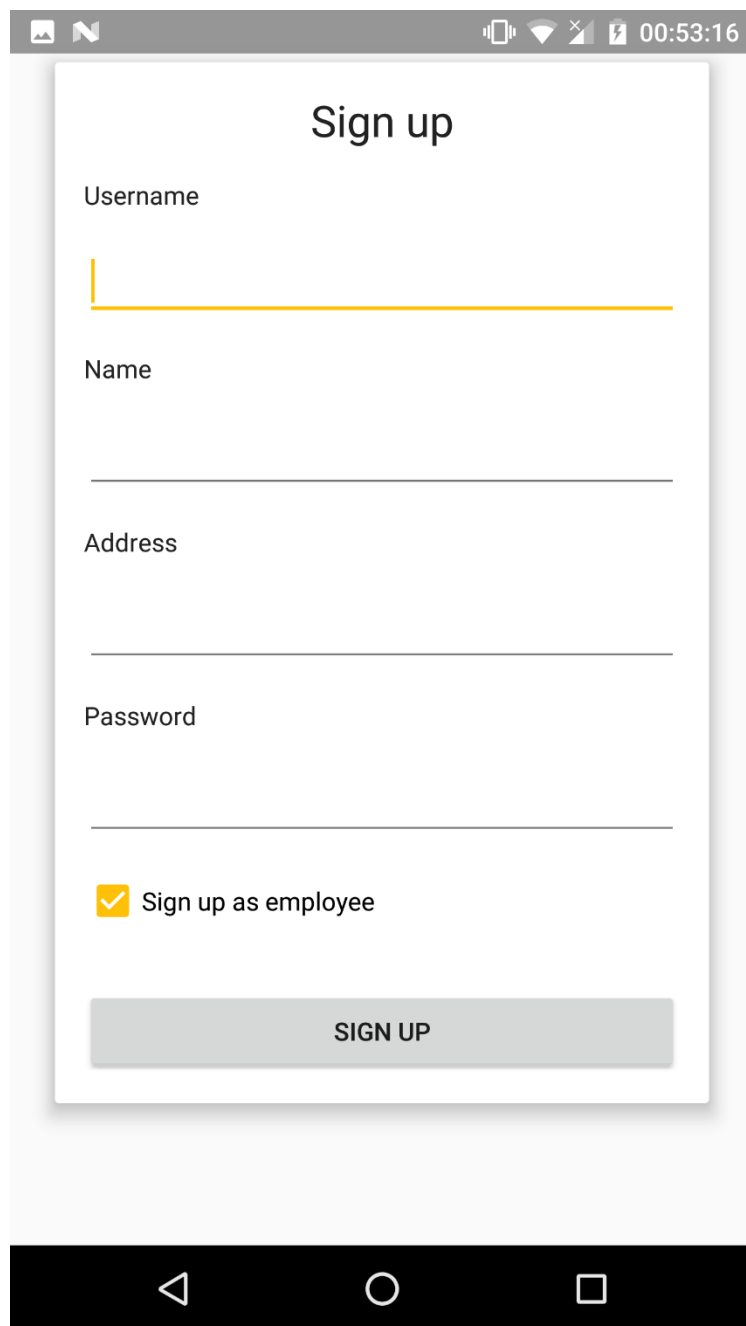
Przy pierwszym włączeniu aplikacji wyświetla się nam ekran logowania. Z tego ekranu mamy też dostęp do aktywności rejestracji. Przy operacji logowania możemy wybrać czy logujemy się jako pracownik czy klient.

The image is a screenshot of a mobile application interface. At the top, there is a dark blue header bar with a hamburger menu icon on the left, the text 'BunchOfTools' in the center, and a vertical ellipsis icon on the right. Below the header, the main content area is white and contains a 'Login' form. The form has a title 'Login' at the top. Below the title are two input fields: 'Username' and 'Password'. Below the 'Password' field is a checkbox labeled 'Login as employee'. At the bottom of the form are two buttons: 'SIGN UP' and 'LOGIN'. The entire form is centered on the screen. At the very bottom of the screen is a black navigation bar with three white icons: a back arrow, a circle, and a square.

Rysunek 144 Ekran logowania

2. Ekran rejestracji

Po przejściu do ekranu rejestracji wyświetlony zostaje formularz który należy wypełnić aby dokonać rejestracji.

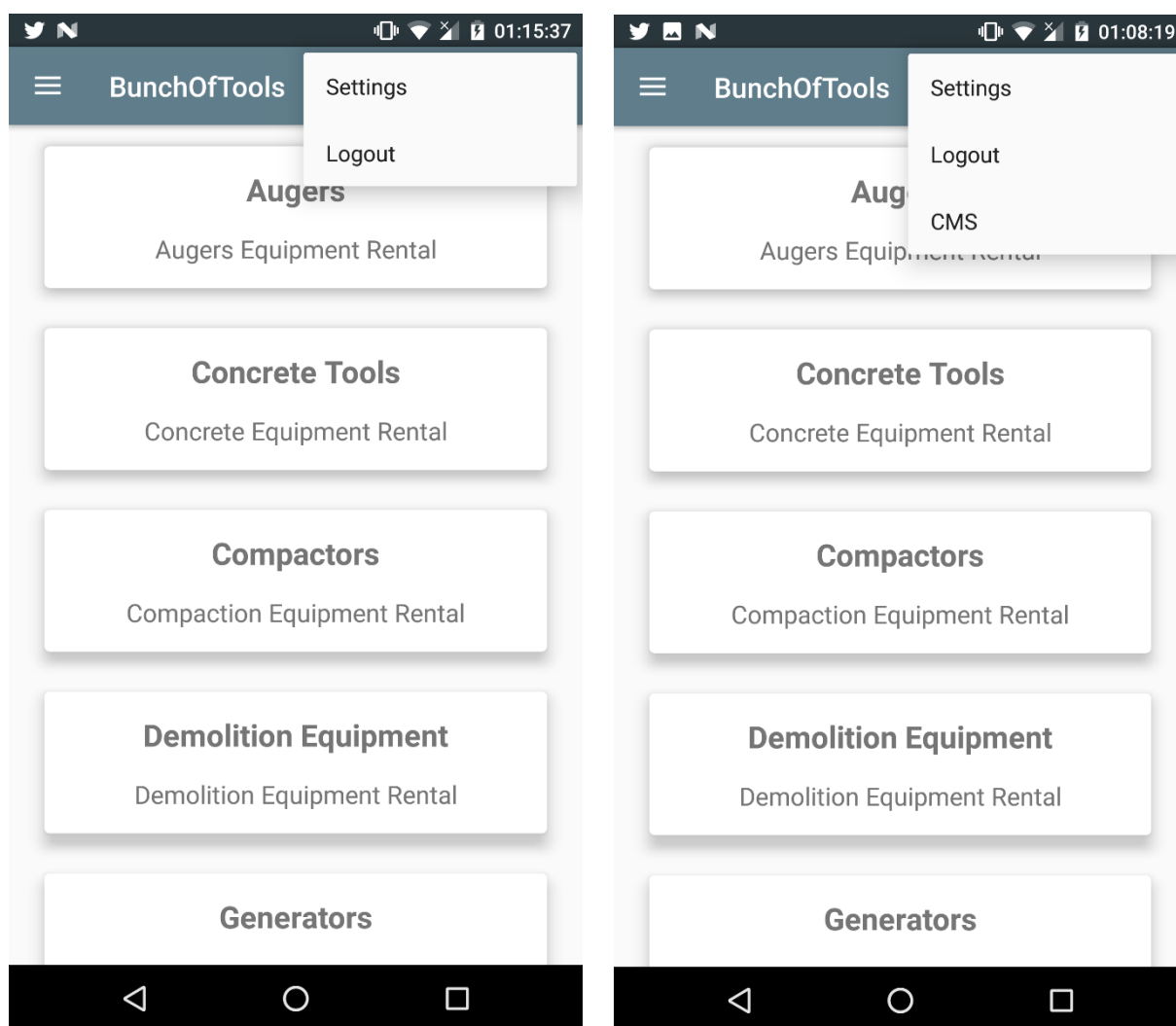


The screenshot shows a mobile application interface for a registration screen. At the top, there is a status bar with icons for signal, Wi-Fi, battery, and the time 00:53:16. Below the status bar is a white card with a light gray shadow. The card has a title 'Sign up' in bold black text. Below the title are four input fields: 'Username' (with a yellow underline), 'Name', 'Address', and 'Password'. Below the 'Password' field is a checkbox with a yellow checkmark and the text 'Sign up as employee'. At the bottom of the card is a gray button with the text 'SIGN UP' in bold black text. The card is set against a light gray background. At the very bottom of the screen is a black navigation bar with three white icons: a triangle, a circle, and a square.

Rysunek 155 Ekran rejestracji

3. Ekran kategorii

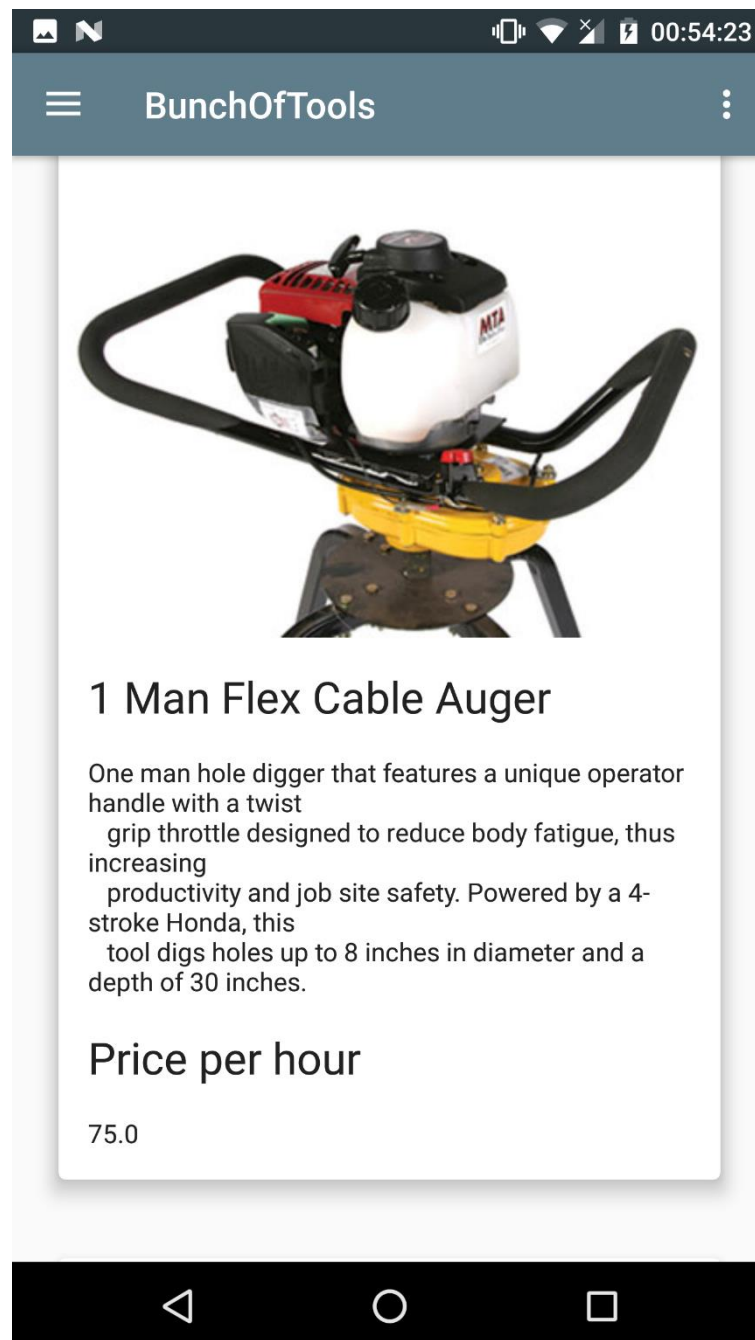
Po zalogowaniu do aplikacji następuje przejście do ekranu kategorii. Nasze uwierzytelnienie (token) zostało zapamiętane w pamięci urządzenia. Gdy następnym razem włączymy aplikację ekranem startowym będzie ekran kategorii, a nie logowania. Jeżeli chcemy się wylogować lub zmienić konto w menu znajdującym się w prawym górnym rogu jest dostępna opcja *Logout*. Wszystkie dane są pobierane z bazy danych poprzez API. Ładowanie danych następuje pomiędzy aktywnościami i zajmuje mniej niż sekundę w pesymistycznym przypadku. Jeżeli zalogowaliśmy się jako pracownik z tego ekranu będziemy mieli również dostęp do CMS.



Rysunek 166, 17 Ekran kategorii po zalogowaniu jako klient / pracownik

4. Ekran modeli

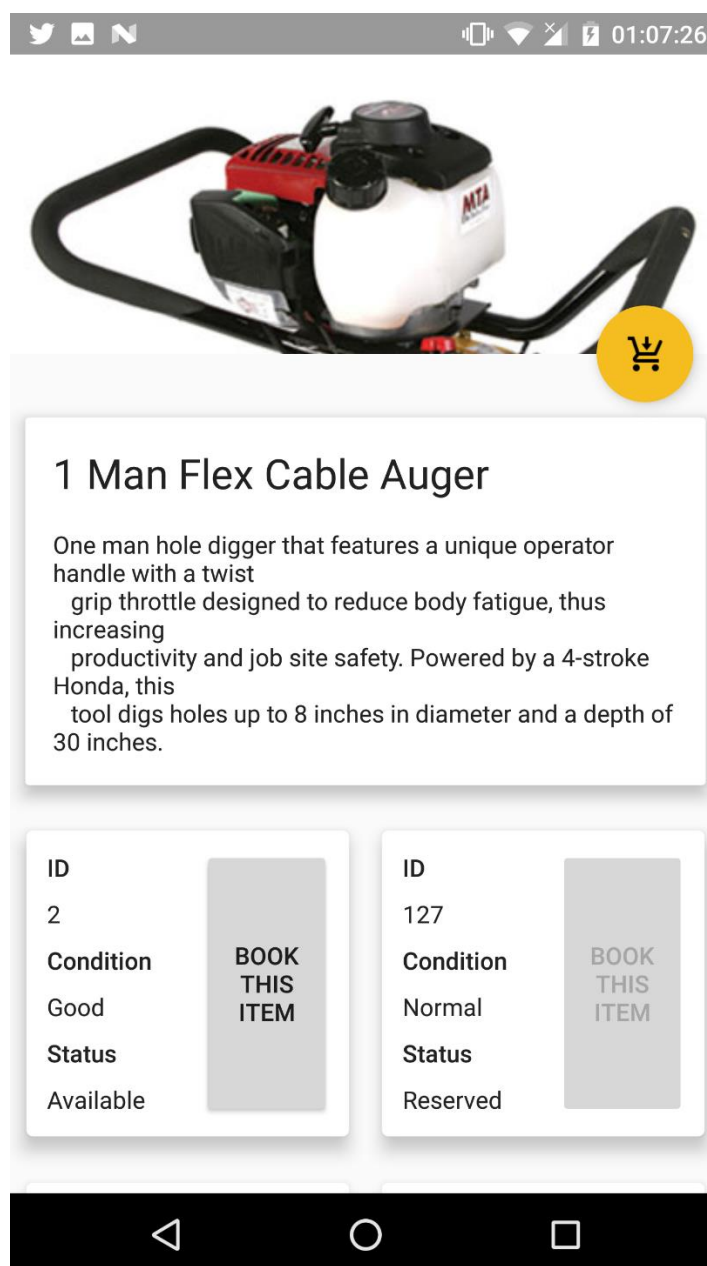
Po wybraniu którejś z kategorii następuje przejście do ekranu modeli. Pobierane i wyświetlane modele są jedynie te których *category_id* odpowiada wybranemu. Aby zapewnić szybkie ładowanie widoku oraz zabezpieczyć się przed `OutOfMemoryError` każde ze zdjęć pobierane jest oddzielnie i tylko w wypadku gdy chcemy je wyświetlić tzn. gdy użytkownik scroluje w dół i wymagane jest pojawienie się kolejnego modelu.



Rysunek 18 Ekran modeli

5. Ekran egzemplarzy

Po wybraniu modelu który nas interesuje przechodzimy do aktywności Item gdzie znajduje się lista pojedynczych egzemplarzy z informacją o ich identyfikatorze, statusie dostępności i zużycia. Te które są dostępne możemy zarezerwować jeżeli jesteśmy zalogowani jako użytkownicy oraz limit zarezerwowanych przedmiotów nie został osiągnięty. Po wykonaniu rezerwacji, dane są odświeżane aby zapewnić spójność pomiędzy danymi w bazie danych a tymi wyświetlanymi w aplikacji.



Rysunek 18 Ekran egzemplarzy

6. Podsumowanie

Projekt został wykonany zgodnie ze wstępnymi założeniami. System wykazuje duży potencjał co do dalszego rozszerzania jego funkcjonalności dzięki optymalnie zaprojektowanej bazie danych oraz wysokiej jakości kodu API jak i aplikacji mobilnej. Jesteśmy bardzo zadowoleni co do jego efektywności. Każda z części systemu działa zgodnie założoną wydajnością dzięki czemu nie występuje zjawisko wąskiego gardła, a cały system jest wysoce responsywny. Praca zespołowa opierająca się na metodykach zwinnych okazała się bardzo dobrym wyborem co znacznie ułatwiło pracę podczas tworzenia poszczególnych komponentów.