JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 14 (2020) 43-47
Wysłane: 2019-10-16
Przyjęte: 2019-10-22

Porównanie systemów mapowania obiektowo relacyjnego greenDao i Room

Maciej Lewinski*
Politechnika Lubelska, Katedra Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Poniższy artykuł skupia się na porównaniu dwóch rozwiązań ORM dla systemu Android: greenDao i Room. Zestawiono sposób użycia tych frameworków. Dokonano pomiaru czasu wykonania operacji wstawienia, aktualizacji, pobierania i usuwania danych z bazy. W graficznej i tabelarycznej formie przedstawiono wyniki. Dokonano ich analizy i zaprezentowano wnioski.

Słowa kluczowe: Android; ORM; greenDao; Room

*Autor do korespondencji.

Adres e-mail: lewinski.maciej1@gmail.com

Comparison of GreenDao and Room ORM Systems

Maciej Lewinski*
Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. This paper focuses on comparison of two ORM libraries for Android: greenDao and Room. Ways of using these frameworks are presented. Time consumption of creating, reading, updating and deleting records are measured. Tables and charts containing results are presented. Finally analysis and conclusions are made.

Keywords: Android; ORM; greenDao; Room

*Corresponding author.

E-mail address: lewinski.maciej1@gmail.com

1. Wstęp

W dzisiejszych czasach urządzenia z systemem Android stały się elementem codziennego życia. Zgodnie ze statystykami prowadzonymi przez *statcounter GlobalStats* 76.23% wszystkich urządzeń mobilnych korzysta z systemu Android (stan na sierpień 2019) [1]. Dla porównania drugi najbardziej popularny system iOS posiadał udział w rynku na poziomie 22.17%.

Właściwie każda aplikacja zasilana jest danymi z bazy danych. Otrzymywane dane muszą być w odpowiedni sposób przystosowane w celu użycia w aplikacji. Proces ten nazywa się mapowaniem obiektowo-relacyjnym (ang. Object-Relational Mapping).

Tworzenie kodu programu, który jest odpowiedzialny za połączenie z bazą danych i wykonywanie na niej operacji jest schematyczne. Pojawiło się wiele frameworków, które wyręczają programistę z pisania powtarzalnych elementów. Użycie ich zmniejsza nakład pracy i ilość kodu minimalizując ryzyko błędów.

Systemy ORM różnią się między sobą. Można je porównywać pod względem ilości linii kodu, który trzeba napisać, żeby je użyć czy wspieranych systemów baz danych. W tym artykule porównywane są dwa z nich greenDao i Room pod względem wydajności i podstawowego sposobu użycia.

2. Przegląd literatury

Użyteczność systemów ORM motywuje autorów do tworzenia prac na ten temat. Jednym z najbardziej popularnych frameworków stworzonych dla języka Java jest Hibernate. W artykule „Efficient Implement of ORM (Object/Relational Mapping) Use in J2EE Framework: Hibernate" [2] poruszany jest problem efektywnego używania tego systemu. Autor przedstawia różne elementy rozwiązania Hibernate i ich sposób użycia. System Android korzysta jednak z bazy SQLite, co wyklucza korzystanie tego systemu. Urządzenia mobilne niosą również dodatkowe wyzwania. W artykule „Understanding the Energy, Performance, and Programming Effort Trade-Offs of Android Persistence Frameworks" porównane zostały rozwiązania ORM skupiając się na wykorzystaniu baterii, obciążeniu bazy danych i czas wykonywania zapytań [3].

Producenci frameworków również tworzą testy sprawności w celu promowania swoich rozwiązań. Firma greenRobot opublikowała porównanie systemów ORM [4]. Zestawiono w nim czasy wykonań operacji CRUD (create, read, update, delete). Zgodnie z tym zestawieniem najszybszy jest framework greenDao. Nie zawarto w nim jednak systemu Room.

Tworząc testy sprawności uruchamiane na telefonach lub tabletach należy mieć na uwadze specyfikę tych urządzeń. W artykule „Benchmarking on Android" [5] poruszone zostały kwestie takie jak oszczędzanie energii, aktualizacja innych

aplikacji i tryb samolotowy oraz ich wpływ na przeprowadzane testy. Należy dbać, aby testy były jak najbardziej obiektywne.

3. Opis obiektu badań

W tej pracy zbadano działanie dwóch frameworków greenDao i Room. Oba dostarczają rozwiązania implementujące mapowanie obiektowo-relacyjne przeznaczone dla systemu Android.

Główną klasą używaną podczas tworzenia aplikacji dla tego systemu jest Activity [6]. Zawiera ona metody, które są wywoływane podczas zmiany stanów aktywności [7]. Są one odpowiedzią na działania użytkownika takie jak naciśnięcie przycisku wstecz czy zamknięcie aplikacji [8]. Aktywności są odpowiedzialne przede wszystkim za zarządzanie widokiem. Za połączenie z bazą danych czy mapowanie obiektowo-relacyjne powinny być odpowiedzialne inne klasy.

GreenDao jest systemem, który używa adnotacji w celu odpowiedniego oznaczenia klas i ich składowych, które powinny zostać użyte [9]. Następujące klasy są podstawą użycia greenDao: DaoMaster, DaoSession, XYZDao, XYZEntity.

DaoMaster jest klasą, która jest punktem dostępowym do systemu greenDao. Ta główna klasa zarządza innymi, które są odpowiedzialne za dostęp do danych danego schematu. Jedną ze składowych klasy DaoMaster jest obiekt bazy danych. Zawiera ona również statyczne metody do tworzenia i usuwania tabel. W postaci klas wewnętrznych dostarcza również implementacje klasy abstrakcyjnej SQLiteOpenHelper: OpenHelper i DevOpenHelper.

DaoSession odpowiedzialne jest za zarządzanie wszystkimi obiektami klas dostępu do bazy, czyli klas DAO (ang. Data Access Object). Za pomocą obiektów tej klasy możemy wywołać również podstawowe metody dostępu do danych.

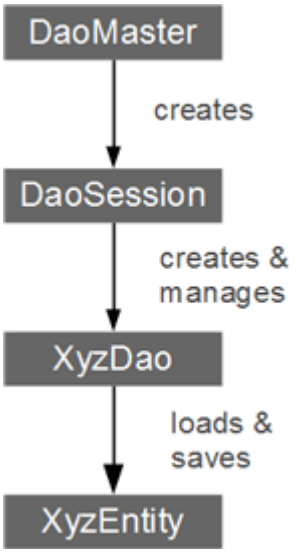
GreenDao dla każdej klasy encji tworzy klasę dostępu do danych. Odpowiedzialne są one za zapisywanie danych i wykonywanie bardziej skomplikowanych zapytań do bazy.

Klasy encji reprezentują tabele w bazie danych. Zazwyczaj jeden obiekt odpowiada jednemu rekordowi. Te klasy oznaczane są adnotacją *@Entity*.

Rysunek 1 ilustruje zależności pomiędzy komponentami systemu greenDao.

System Room tworzy poziom abstrakcji używany do łatwego dostępu do bazy SQLite [10]. Pozwala to na odseparowanie logiki biznesowej aplikacji od części odpowiedzialnej za pobieranie i zapisywanie danych. Jest to oficjalnie zalecany framework dla systemu Android do pracowania z lokalną bazą danych.

System ORM Room składa się z trzech podstawowych elementów, a mianowicie *Database*, *Entity* i *Dao*.



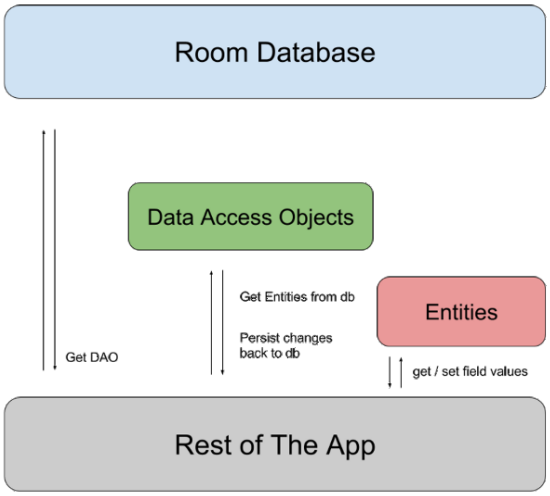
Rys. 1. Zależności między klasami systemu greenDao [9]

Klasa oznaczona adnotacją *@Database* i rozszerzająca *RoomDatabase* jest punktem wyjściowym do użycia systemu Room. Dostarcza ona połączenie do bazy danych. Obiekt tej klasy można uzyskać wywołując statyczną metodę *databaseBuilder()* klasy *Room*.

Klasy rodzaju *Entity* są definicjami tabel w bazie danych. Oznaczane są adnotacją *@Entity*. Ich instancje reprezentują wiersze danych.

W celu użycia obiektów DAO programista musi zdefiniować interfejs oznaczony adnotacją *@Dao*. System Room sam wygeneruje klasę go implementującą.

Rysunek 2 ilustruje zależności między komponentami systemu Room.



Rys. 2. Zależności między komponentami frameworka Room [10]

4. Metody badawcze

W celu zbadania szybkości działania wybranych systemów ORM zbudowano aplikację wykonującą te same operacje w bazie danych za pomocą różnych frameworków.

W tabeli 1 przedstawiono użyte wersje badanych systemów ORM.

Tabela 1. Wersje użytych systemów ORM

Nazwa systemu	Wersja
Room	2.1.0
greenDao	3.2.2

4.1. Wstawianie rekordów

Pierwszy test polegał na pomiarze czasu wstawiania rekordów do bazy. Klasy reprezentujące encje zawierały pola wszystkich typów prostych i odpowiadających im typów referencyjnych. Przed rozpoczęciem wstawiania tworzono listę wypełnioną odpowiednią ilością gotowych do wstawienia rekordów. Pola prymitywnych typów liczbowych były inicjalizowane minimalnymi wartościami, jakie mogły zawierać, pola referencyjne zainicjalizowano wartością *null*, a pole typu *boolean* zawierało wartość *false*. Przed startem pomiaru inicjalizowano również połączenie z bazą danych. Wypełniana tabela była pusta.

4.2. Aktualizowanie danych w bazie

Kolejną operacją, która była mierzona było aktualizowanie danych w bazie. Użyto tabeli, która została zastosowana do pomiaru czasu wstawiania. Użyto takiej liczby rekordów, jaka została dodana w poprzedzającym badaniu. Przed rozpoczęciem pomiaru wypełniono danymi listę obiektów klas reprezentujących encje. Pola prymitywnych typów liczbowych były inicjalizowane maksymalnymi wartościami, jakie mogły zawierać, pola referencyjne zainicjalizowano wartością *null*, a pole typu *boolean* zawierało wartość *true*.

4.3. Pobieranie rekordów z bazy

Ten test polegał na pomiarze czasu pobierania uprzednio zaktualizowanych danych. Indeksy wierszy były kolejnymi liczbami naturalnymi zaczynając od liczby jeden. Wykonywano pojedyncze zapytania do bazy danych. Podając wartość *id* rekordu wczytywano go do obiektu. Następnie wywoływano wszystkie gettery, aby zapobiec ewentualnemu odroczonemu wczytywaniu.

4.4. Usuwanie danych z bazy

W tym teście mierzono czas pojedynczego usuwania rekordów z bazy. Tabela, z której korzystano zawierała dane, które były pobierane w poprzednim badaniu. Dla każdej wartości *id* z osobna wywoływano metodę usuwania.

5. Wyniki badań

Wszystkie pomiary wykonano dla 250, 500, 1 000, 2 000, 5 000, 10 000, 20 000 rekordów. Testy w każdej konfiguracji powtórzono pięciokrotnie. Użytym urządzeniem był telefon Lenovo C2, posiadający 1 GB RAM. Zainstalowanym oprogramowaniem był system Android 6.0.

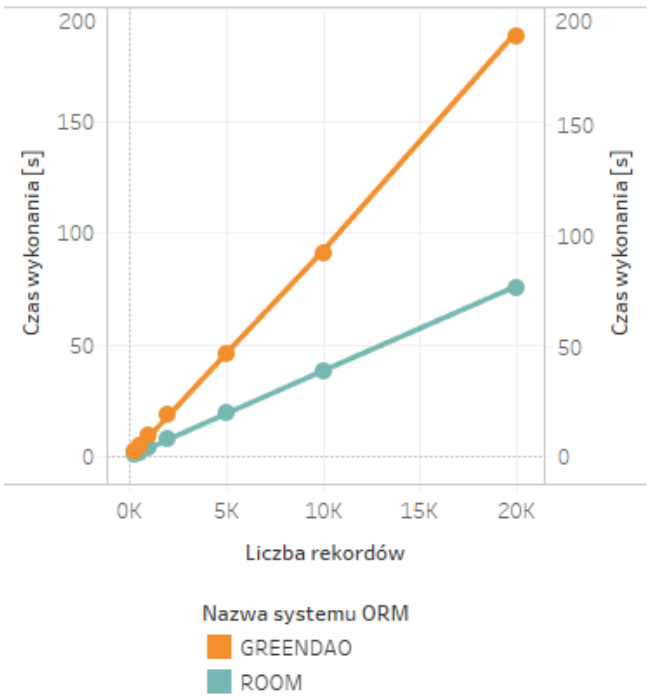
5.1. Wstawianie rekordów

Przedstawiony na rysunku 3 wykres bardzo wyraźnie wskazuje na liniową zależność między czasem wykonywania a liczbą rekordów. Jasno pokazuje on, że system greenDAO działał znacznie wolniej niż framework Room.

W tabeli 2 ujęto dokładne czasy wykonywania. Bez względu na liczbę rekordów system Room wykonywał tę operację przynajmniej dwa razy szybciej niż greenDAO.

Ostatecznie oficjalny framework dla systemu Android potrzebował średnio 0,3879 ms na wstawienie jednego rekordu, podczas, gdy ten publikowany przez greenRobot wymagał 0,9422 ms. Tak więc ten drugi potrzebował o 143 % więcej czasu.

Wstawienie 250 rekordów zajęło systemowi Room średnio 1,1 s. Jest to czas dłuższy niż 1 s, która jest graniczna wartością zapewniajaca użytkownikowi poczucie płynnego działania aplikacji. Oznacza to, że oczekiwanie na wstawienie takiej ilości rekordów może powodować dyskomfort.



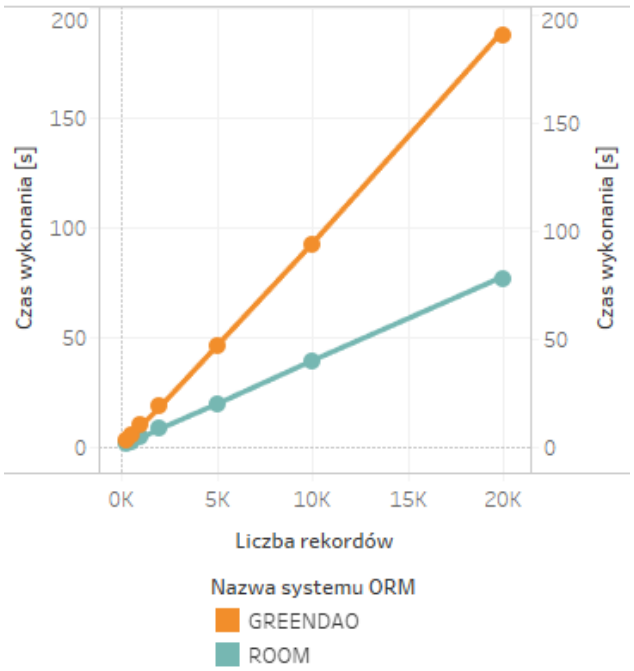
Rys. 3. Wykres zależności czasu wykonania operacji wstawiania od liczby rekordów

Tabela 2. Średni czas wykonania operacji wstawienie w zależności od liczby rekordów i systemu ORM

Liczba rekordów	Room	greenDao
250	1,1 s	2,4 s
500	2,2 s	4.9 s
1 000	4,0 s	9.6 s
2 000	8,1 s	18,8 s
5 000	19,6 s	46,7 s
10 000	38,8 s	92,5 s
20 000	76,5 s	190,3 s

5.2. Aktualizowanie rekordów

Czas aktualizacji jednego rekordu za pomocą systemu Room wynosił średnio 3,931 ms. Jest to wartość o 58 % niższa od wyniku frameworka greenDao, który potrzebował ok 9,415 ms. Tabela 3 pokazuje, że czasochłonność operacji operacji aktualizacji jest prawie taka jak operacji wstawiania. Posiada również wręcz identyczna liniową charakterystykę zależności od ilości rekordów, co widoczne jest na umieszczonym na rysunku 4 wykresie.



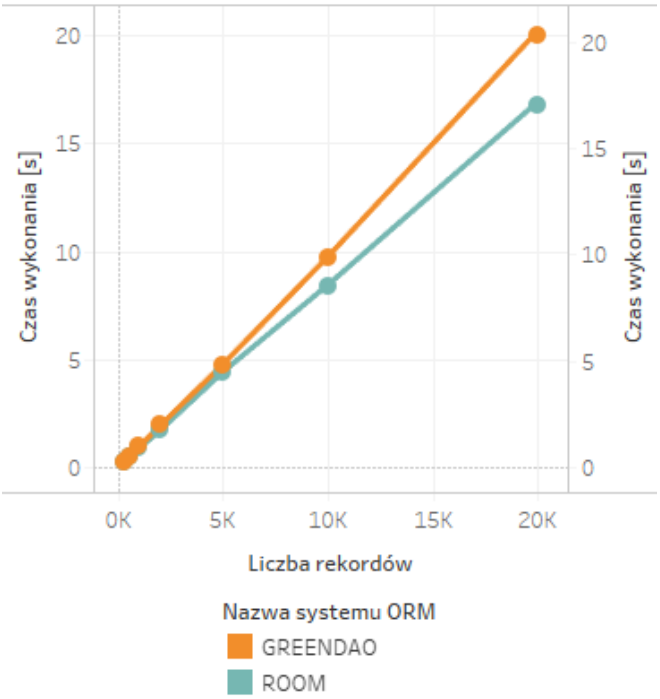
Rys. 4. Wykres zależność czasu wykonania operacji aktualizacji od liczby rekordów

Tabela 3. Średni czas wykonania operacji aktualizacji w zależności od liczby rekordów i systemu ORM

Liczba rekordów	Room	greenDao
250	1,1 s	2,6 s
500	2,0 s	4,8 s
1 000	4,1 s	9,5 s
2 000	8,3 s	18,3 s
5 000	19,7 s	46,3 s
10 000	39,4 s	93,0 s
20 000	77,8 s	190,2 s

5.3. Pobieranie rekordów

Wyniki pomiarów pokazują, że pobieranie rekordów z bazy danych jest operacją znacznie szybszą niż ich wstawianie czy aktualizacja. W tej kategorii system Room okazał się być również szybszy. Średnio pobranie jednego rekordu zajmowało temu frameworkowi 0,8601 ms. System greenDAO potrzebował 16 % więcej czasu, a mianowicie 0,9988 ms. Tak mała różnica w szczególności dla bardzo krótkich okresów czasu jest dla użytkownika praktycznie niezauważalna. Dokładne wartości zostały zaprezentowane w tabeli 4. Wykres przedstawiony na rysunku 5 pokazuje liniową zależność między czasem wykonania operacji pobierania a ilością rekordów.



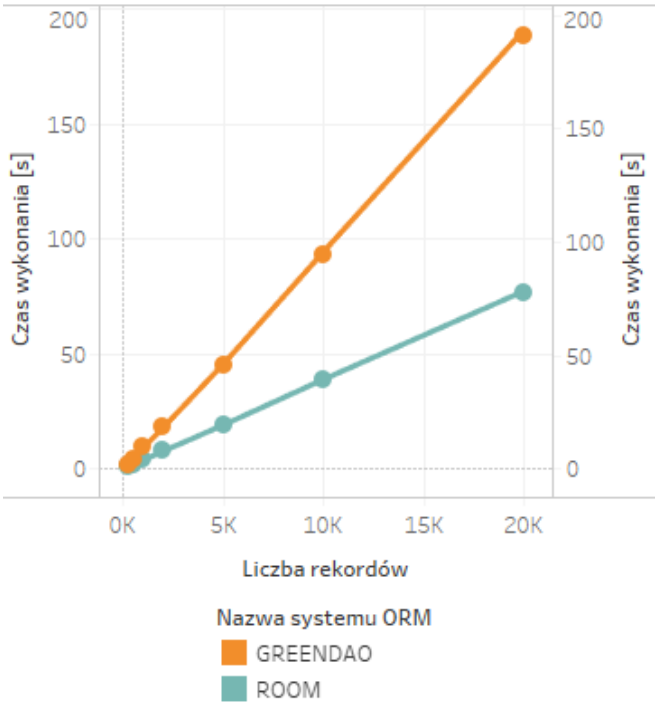
Rys. 5. Wykres zależności czasu wykonania operacji pobierania od liczby rekordów

Tabela 4. Średni czas wykonania operacji pobierania w zależności od liczby rekordów i systemu ORM

Liczba rekordów	Room	greenDao
250	0,22 s	0,27 s
500	0,49 s	0,52 s
1 000	0,89 s	0,99 s
2 000	1,75 s	1,97 s
5 000	4,46 s	4,82 s
10 000	8,50 s	9,82 s
20 000	17,01 s	20,31 s

5.4. Usuwanie rekordów

Umieszczony na rysunku 6 wykres pokazuje liniową zależność między czasem usuwania a liczbą rekordów. Wiedząc, że taka jest relacja można szacować czas wykonywania dla innej ilości danych. Pod względem usuwania danych szybszy ponownie okazał się system Room. Średni czas usuwania jednego rekordu przez ten framework wyniósł 3,913 ms. System greenDao potrzebował średnio na jeden rekord 9,468 ms, czyli o 142 % więcej. Tabela 5 pokazuje, że różnice w wykonywaniu tej operacji są znaczące.



Rys. 6. Wykres zależności czasu wykonania operacji usuwania od liczby rekordów

Tabela 5. Średni czas wykonania operacji usuwania w zależności od liczby rekordów i systemu ORM

Liczba rekordów	Room	greenDao
250	1,1 s	2,4 s
500	2,0 s	4,7 s
1 000	4,2 s	9,6 s
2 000	8,0 s	18,4 s
5 000	19,4 s	46,0 s
10 000	39,2 s	94,8 s
20 000	77,6 s	191,1 s

6. Analiza wyników i wnioski

Użycie obu frameworków jest bardzo podobne. Zawarcie ich w projekcie polega na dodaniu odpowiednich zależności w pliku Gradle. Zarówno system greenDao jak i Room używa adnotacji w celu oznaczenia mapowanych klas.

Znacząca różnica występuje jednak w sposobie użycia klas DAO. System Room umożliwia pisanie zapytań SQL, które są walidowane w czasie kompilowania projektu. greenDao pozwala na pisanie tzn. "surowych" zapytań lecz niestety nie są one sprawdzane podczas kompilacji. Typowym podejściem wspieranym przez system dostarczony przez greenRobot jest używanie budowniczego zapytań. Taka możliwość nie jest dostarczana przez system Room.

Framework dostarczany przez twórców systemu Android jest więc bardziej odpowiedni dla osób, które głównie korzystają z pisania zapytań w języku SQL. Walidacja podczas kompilacji znacząco ułatwia znajdowanie i naprawę błędów. Programiści, którzy nie znają SQL znaczanie szybciej odnajda się w używaniu systemu greenDAO, gdyż używanie klasy *QueryBuilder* będzie dla nich wygodniejsze.

Jeżeli dla tworzonej aplikacji osiągi czasowe są decydujące, to użycie systemu Room będzie zdecydowanie lepsze. Oferuje on znacznie szybsze wykonywanie operacji na bazie danych. Czasy pobierania były zbliżone, lecz wstawianie, aktualizacja i usuwanie danych było co najmniej dwa razy szybsze.

Literatura

[1] https://gs.statcounter.com/os-market-share/mobile/worldwide [30.09.2019]

[2] Chuanlong Xia, Guangcan Yu, Meng Tang: Efficient Implement of ORM (Object/Relational Mapping) Use in J2EE Framework: Hibernate. 2009 IEEE International Conference on Computational Intelligence and Software Engineering

[3] J. Pu, Z. Song, E. Tilevich, Understanding the Energy, Performance, and Programming Effort Trade-Offs of Android Persistence Frameworks, 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

[4] http://greenrobot.org/android/android-orm-performance-2016 [14.10.2019]

[5] http://greenrobot.org/android/benchmarking-on-android/ [11.10.2019]

[6] Allen Grant, Beginning Android, Apress, 2015

[7] MacLean Dave, Komatineni Satya, Allen Grant, Pro Android 5, Apress, 2015

[8] https://developer.android.com/guide/components/activities/activity-lifecycle [05.10.2019]

[9] http://greenrobot.org/greendao/documentation/introduction/ [04.10.2019]

[10] https://developer.android.com/training/data-storage/room [04.10.2019]