|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | **Wydział Matematyki Stosowanej** | |
|  |  |  |  | **Politechnika Śląska** |
|  |  |  |  |  |
| **Kierunek:** |  |  |  | Informatyka |
| **Stopień studiów:** | | |  | pierwszy |
| **Semestr:** |  |  |  | V |
| **Przedmiot:** |  |  |  | Aplikacje mobilne dla systemu Android |
| **Prowadzący zajęcia:** | | |  | dr inż. Adam Zielonka |
|  |  |  |  | **Dokumentacja techniczna** |
|  |  |  | **realizacji projektu zespołowego** | |
|  |  |  |  |  |
|  |  |  |  | **Zespół projektowy | Podział pracy** |
|  |  |  |  | |
| **Lider zespołu:** |  |  | Kamil Czaja [kamil.czaja94@gmail.com](mailto:kamil.czaja94@gmail.com) | Składowanie i obsługa danych | |
| **Wykonawca 1:** |  |  |  | |
| **Wykonawca 2:** |  |  |  |  |
| **Wykonawca 3:** |  |  | Norbert | Wieczorek [*wieczoreknorbert93@gmail.com*](mailto:wieczoreknorbert93@gmail.com)|Gui, Funkcjionalność |
|  |  |  |  | **Projekt** |
|  |  | | | |
| **Tytuł projektu:** |  | | | |
| **Repozytorium:** |  | | | |
| **Cel projektu:** |  |  |  |  |
|  |  | | | |
|  |  | | | |
|  |  |  |  |  |
| **Opis projektu:** |  |  |  |  |
|  |  | | | |
|  |  | | | |
|  |  | | | |
|  |  | | | |
|  |  |  |  |  |
|  |  |  |  | **Użyte narzędzia** |
|  |  | | | |
| Nazwa: | Android Studio 2.2.3 | | | |
| Licencja: | Personal (free) | | | |
| Nazwa: |  | | | |
| Licencja |  | |  |  |

Spie treści:

1. Cel projektu.......................................................................................................................1
2. Podział prac.......................................................................................................................1
3. Założenia do projektu.......................................................................................................1
4. Struktura danych..............................................................................................................1

4.1

4.2

...

1. Struktura aplikacji...........................................................................................................1

5.1.Menadżer fragmentów (Fragment Holder)..............................................................1

5.2.Fragmenty główne....................................................................................................1

5.2.1. Baza Danych…………………………………………………………….1

5.2.2. StartWindow(Home)................................................................................1

5.2.3. ListWindow(Workout).............................................................................1

5.2.4. ListWindow(Meals).................................................................................1

5.2.5. ProfilWindows(Profil)..............................................................................1

5.3. Kontakt pomiędzy fragmentami...............................................................................1

5.4.Fragmenty pomocnicze............................................................................................1

5.4.1.TopControlPanel......................................................................................1

5.4.2. CalendarWindow......................................................................................1

5.4.2. InputWindow............................................................................................1

5.4.3.ExerciseWindow......................................................................................1

1. Cel projektu:
2. Podział prac

Norbert Wieczorek:

- Menadżer Fragmentów

- Strukturwa poczatkowa aplikacji

- Listowanie cwiczen i treningów z bazy danych

- Interfejs dodawania nowych cwiczen i treningów

- Wyświetlanie szczegółów na temat ćwiczeń i treningów

- Przeszukiwanie danych i wyswietlanie w liscie (niezrealizowano)

- Interfejs zliczający cwiczenia (niezrealizowano)

Kamil Czaja

- Ciągłe modyfikacje modelu aplikacji oraz modelu bazodanowego

- Architektura dostępowa do bazy danych

- Repozytoria dla każdego typu obiektu

- Metody pozwalające na prace reszty zespołu bez konieczności obsługi czegokolwiek w SQL

- Metoda wypełniające baze danych (dla Debug)

- Drobne prace w UI

1. Założenia do projektu
2. Struktura danych
3. Struktura aplikacji:
   1. Baza danych:

Baze danych wykonanu w oparciu o silnik bazodanowy SQLite. W folderze assets znajduje się aktualna wersja bazy danych wraz z podstawowym wypełnieniem.

Aplikacja podczas uruchamiania wykrywa czy posiada baze danych oraz czy jest ona w najnowszej wersji. W przypadku stwierdzenia nie istnienia pliku bazodanowego aplikacja przekopiowuje strukturę i dane z bazy zawartej w folderze assets.

Dostęp do danych zawartych w bazie obywa się poprzez model repozytoriów. Każdy model używany w aplikacji posiada swoje repozytorium wewnątrz którego znajduja się metody umozliwiające dostęp do danych.

Np. Model UserInformation posiada swoje repozytorim InformationRepository w którym znajdują się wszystkie bazodanowe metody tego obiektu tj:

- putInformationData które w argumencie przyjmuje wypełniony model, następnie wpisując go do bazy danych.

- getInformationData Która zwraca aktualny pełen model odnośnie informacji o użytkowniku.

Pozostałe modele posiadają swoje analogiczne repozytoria.

* 1. Pobieranie danych z bazy zostanie omówione na przykładzie InformationRepository

**try**{  
 SQLiteDatabase DB = db.getReadableDatabase();  
 String[] coloumns = {InformationTableDetails.***COLUMN\_WAGA***, InformationTableDetails.***COLUMN\_WZROST***,InformationTableDetails.***COLUMN\_KALORIE***,  
 InformationTableDetails.***COLUMN\_POZIOM***,InformationTableDetails.***COLUMN\_TLUSZCZ***,  
 InformationTableDetails.***COLUMN\_WEGLOWODANY***,InformationTableDetails.***COLUMN\_BIALKO***,InformationTableDetails.***COLUMN\_CEL***,InformationTableDetails.***COLUMN\_PLEC***,  
 InformationTableDetails.***COLUMN\_BUDOWA***,InformationTableDetails.***COLUMN\_AGE***,InformationTableDetails.***COLUMN\_ID***};  
  
 Cursor CR = DB.query(InformationTableDetails.***TABLE\_NAME***, coloumns,**null**,**null**,**null**,**null**,InformationTableDetails.***COLUMN\_ID***+**" DESC"**);  
  
 CR.moveToFirst();  
 UserInformation userData= **new** UserInformation(Float.*parseFloat*(CR.getString(0)),(Float.*parseFloat*(CR.getString(1))),  
 (Float.*parseFloat*(CR.getString(2))),(Float.*parseFloat*(CR.getString(3))),(Float.*parseFloat*(CR.getString(4))),  
 (Float.*parseFloat*(CR.getString(5))),(Float.*parseFloat*(CR.getString(6))),CR.getString(7),CR.getString(9),  
 Integer.*parseInt*(CR.getString(8)),Integer.*parseInt*(CR.getString(10)));  
  
 Log.*d*(**"DataBase operations"**,**"Data recievied from DB"**);

DB.close();

**return** userData;  
}  
**catch** (Exception ex){  
 **return new** UserInformation();  
}

W pierwszej kolejności musimy uzyskac uchwyt do bazy daych, z przekazanego w argumencie obiektu DatabaseHelper. Dzięki temu zabiegowi możemy utworzyć base w trybie zapisu lub odczytu.

Następnie musimy wskazać które dane chcemy z bazy pobrac. Każde repozytorum posaida klase statyczna która zawiera stałe z nazwami kolumn w tabeli. To rozwiązanie zaoszczędza sporo czasu, nie trzba pamiętać dokładnych nazw kolumn z bazy.

Po stworzeniu tablicy kolumn wykonujemy zaputanie które zwraca nam Coursor. Na podstawie otrzymanych danych jest tworzony obiekt który następnie jest zwracany. W przypadku błędu jest zwracany pusty obiekt.

* 1. Wpisywanie danych z bazy zostanie omówione na przykładzie InformationRepository

SQLiteDatabase DB = db.getWritableDatabase();  
ContentValues content = **new** ContentValues();  
content.put(InformationTableDetails.***COLUMN\_WAGA***, data.getWeight());  
content.put(InformationTableDetails.***COLUMN\_WZROST***, data.getHeight());  
content.put(InformationTableDetails.***COLUMN\_KALORIE***, data.getCal());  
content.put(InformationTableDetails.***COLUMN\_POZIOM***, data.getActivityLvl());  
content.put(InformationTableDetails.***COLUMN\_TLUSZCZ***, data.getFat());  
content.put(InformationTableDetails.***COLUMN\_WEGLOWODANY***, data.getCarb());  
content.put(InformationTableDetails.***COLUMN\_BIALKO***, data.getProtein());  
content.put(InformationTableDetails.***COLUMN\_CEL***, data.getTarget());  
content.put(InformationTableDetails.***COLUMN\_PLEC***, data.getSex());  
content.put(InformationTableDetails.***COLUMN\_BUDOWA***, data.getBodyType());  
content.put(InformationTableDetails.***COLUMN\_AGE***, data.getAge());  
  
  
**long** success = DB.insert(InformationTableDetails.***TABLE\_NAME***,**null**,content);  
Log.*d*(**"DataBase operations"**, **"Row inserted to informacje"**);  
DB.close();

W pierwszej kolejności tak jak w przypadku pobierania danych tworzymy uchwyt do bazy danych. Następnie tworzymy obiekt ContentValues któremy przypisujemy (klucz,wartość) gdzie klucz jest nazwa kolumny tabeli a wartość danymi pobranymi z otrzymanego obiektu do wstawienia do bazy. Następnie wykonujemy inserta. Dla Debug aplikacja loguje w konsoli swoje działania związane z bazą.

* 1. Menadżer fragmentów- posiada w sobie obiekty wszystkich fragmentów które zostały użyte w aplikajci posiada metode getFragment która przyjmije index fragmentu który chcemy wydobyć z menadżera .

|  |
| --- |
| public void fragmentReplace(intwhere, inttag){ transaction = manager.beginTransaction(); transaction.replace(where, holder.getFragment(getTag(tag))); transaction.commit(); } |

\*metoda getTag przerabia int tag który jest w postacji (R.integer.startWindow) na postac numeryczna w tym przypadku 1

W tym fragmencie kodu pokazujemy metode ktora znajduje się w MainActivity i podmienia w miejsce int „where” (w postaci R.id.\*miejsce\*) na inny fragment int tag (R.integer.\*nazwa fragmentu\*)

Aby skontaktować sie z tą metodą użyliśmy formuły „((MainActivity)getActuvity).\*”

|  |
| --- |
| create\_new.setOnClickListener(new View.OnClickListener() { @Override public void onClick(View v) {  ((MainActivity) getActivity()).setInputType(R.integer.*Training*); ((MainActivity) getActivity()).fragmentReplace(R.id.*Main*,R.integer.*inputWindow*); } }); |

* 1. Fragmentygłówne