Uniwersytet Marii Curie-Skłodowskiej w Lublinie

Wydział Matematyki, Fizyki i Informatyki

Instytut Informatyki

Marcin Gawski

Nr indeksu: 218220

Wykorzystanie zdalnego magazynu danych w aplikacjach mobilnych

Praca licencjacka

przygotowana w Instytucie Informatyki

promotor: prof. dr hab. Zdzisław Łojewski

Lublin 2013

Spis treści

Wstęp

Rynek urządzeń mobilnych jest obecnie najdynamiczniej rozwijającym się sektorem branży IT. Sprzedaż zwykłych pecetów z roku na rok maleje. Smartfony i tablety stopniowo wypierają tradycyjne komputery czy laptopy. Coraz większy popyt na urządzenia mobilne i oczekiwania stawiane przez ich użytkowników, sprawiają, ze producenci oprogramowania coraz częściej skupiają się na platformach Android czy iOS. Same urządzenia mobilne niosą ze sobą niezrównany potencjał, ogrom nowych, wcześniej niedostępnych możliwości.

Urządzenia mobilne to nie tylko przenośne centrum rozrywki ale często również potężne narzędzie wspierające różnego rodzaju procesy biznesowe. Stały dostęp do internetu, poczty elektronicznej, sieci społecznościowych, notowań giełdowych, telewizja strumieniowa czy bankowość mobilna to niektóre z ważniejszych możliwości jakie dają nam współczesne urządzenia przenośne. Powyższe rodzaje oprogramowania łączy jedna wspólna cecha – centralny magazyn danych aplikacji fizycznie nie może znajdować się bezpośrednio w urządzeniu. Oczywiście wszystkie platformy mobilne posiadają natywną obsługę lokalnych baz danych. Jednak dla powyższych przypadków jej zastosowanie w praktyce jest często bardzo ograniczone lub nie jest ona wykorzystywana w ogóle. Rozważając przykład aplikacji obsługującej rachunek bankowy, bardzo szybko dochodzimy do wniosku, że przechowywanie jakichkolwiek danych związanych z chociażby stanem konta czy numerem karty kredytowej byłoby skrajnie niebezpiecznym rozwiązaniem. Ryzyko potęguje fakt, że urządzenia mobilne są obecnie bardzo narażone na kradzież, można je też łatwo zgubić. Należy więc zadbać o to, aby wrażliwe informacje nie dostały się w niepowołane ręce. W tym miejscu pojawia się wspólny problem – w jaki sposób przeprowadzić mechanizm bezpiecznej synchronizacji zewnętrznego magazynu danych z aplikacją umieszczoną na setkach różnych urządzeń mobilnych, zapewniając przy tym wysoką funkcjonalność i atrakcyjność aplikacji.

W niniejszej pracy zostanie opracowany i przedstawiony jeden ze sposobów realizacji bezpiecznego mechanizmu wymiany danych pomiędzy urządzeniem mobilnym a zdalnym magazynem danych. W pierwszej kolejności opisane zostaną możliwości lokalnego przechowywania danych bezpośrednio w urządzeniach. Następnie rozważone zostaną aspekty, które należy wziąć pod uwagę podczas

projektowania zdalnego magazynu danych. Projektowaniu i implementacji towarzyszyć będzie przegląd możliwości i dostępnych technologii, zakończony wyborem najlepiej realizującym postawione założenia. Wszystko zostanie zaimplementowane na przykładzie aplikacji ***Diabetes Diary*** utworzonej na potrzeby niniejszej pracy. Będzie się ona składała z trzech osobnych aplikacji współdzielących jedno źródło danych – jedną z nich będzie program napisany na platformę mobilna Android, dwie pozostałe będą aplikacjami internetowymi.

1. Charakterystyka dziedziny problemu

Trudności związane z cukrzycą, wymagają od pacjenta stałej kontroli poziomów cukru we krwi oraz ilości przyjmowanej insuliny. Diabetycy często prowadzą papierowy dziennik w którym zapisują pomiary cukrów w ciągu dnia oraz ilości podanej insuliny. Taki dzienniczek ułatwia pacjentom oraz lekarzom przeprowadzanie zmian w leczeniu. Dzięki takim informacjom można odpowiednio skorygować pory przyjmowania posiłków, ilości przyjmowanej insuliny bądź rodzaje posiłków.

1.1. Użytkownicy systemu

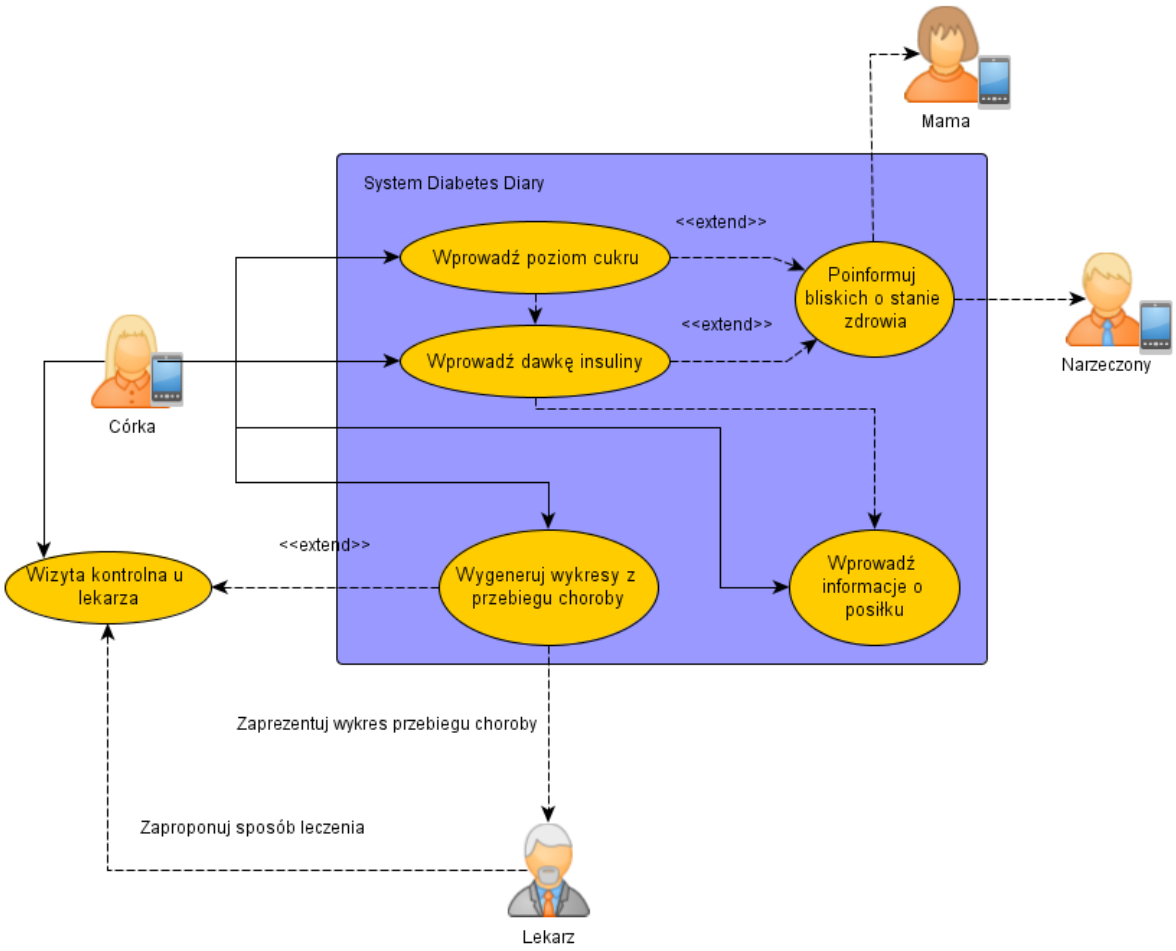
Grupą docelowych użytkowników aplikacji będą więc osoby chorujące na cukrzycę oraz ich rodziny. Program będzie gromadził dane o przebiegu choroby pacjenta. W szczególności będą to następujące informacje:

- Rodzaje przyjmowanej insuliny
- Używany glukometr
- Pomiary cukru w ciągu dnia
- Liczba podanych jednostek insuliny w ciągu dnia
- Liczba i skład posiłków w ciągu dnia

Z punktu widzenia osoby chorej, program umożliwi lepszą kontrolę nad chorobą. Usprawni on również konsultacje lekarskie poprzez możliwość dostarczenia lekarzowi wykresów z postępowania choroby z wybranego okresu. Tak przedstawione wykresy ułatwią odczytanie i interpretację zgromadzonych danych, co jest kluczowe przy ustalaniu dalszego leczenia. Tylko nieliczne gabinety lekarskie posiadają możliwość odczytu pomiarów cukrów bezpośrednio z glukometru pacjenta. Zazwyczaj dwa tygodnie przed planowaną wizytą lekarską, pacjenci muszą samodzielnie spisywać w formie tabeli pomiary glukozy i informacje o iniekcjach insuliny w papierowym dzienniczku. Taki raport mieści się na kilkunastu stronach dziennika. Podczas wizyty, lekarz analizuje przedstawione dane. Forma ich zapisu i prezentacji jest jednak mało czytelna co utrudnia wyciągnięcie odpowiednich wniosków. Możliwość przedstawienia lekarzowi wszystkich danych w przejrzystej formie na wykresie, jest więc niezwykle cenną funkcją. Po analizie zgromadzonych

informacji, aplikacja będzie również starała się sugerować wielkości i częstotliwości przyjmowania insuliny.

Cukrzyca jest chorobą, która angażuje nie tylko osobę chorą, ale również bliskie jej osoby. Z wywiadu środowiskowego dowiedzieć się można, że zdecydowana większość osób chorych na cukrzycę posiada w swoim otoczeniu osoby bliskie, które są zaangażowane w przebieg postępowania choroby. Aplikacja mobilna będzie więc umożliwiała osobom bliskim wgląd do wyników pacjenta. Szczególnie ważną informacją w przebiegu choroby, jest poziom cukru i ilość przyjętej insuliny. Pomiary glukozy i iniekcje insuliny wykonywane są kilkakrotnie w ciągu dnia. Osoby bliskie będą natychmiast informowane o wprowadzonych przez pacjenta pomiarach. Z założenia wszystkie dane mają być osiągalne zarówno przez telefon komórkowy jak i za pośrednictwem serwisu internetowego.



Ilustracja 1: Diagram przypadków użycia systemu Diabetes Diary

1.1.1. Pacjent

Za pomocą aplikacji osoba chora na cukrzycę będzie mogła w szybki oraz prosty sposób dodawać informacje o pomiarach cukru, ilości przyjętej insuliny oraz o porach i składzie posiłków. Pacjent będzie mógł również modyfikować i usuwać dane. Zgromadzone informacje będą osiągalne zarówno z poziomu urządzenia mobilnego (np. telefonu) jak również przez stronę internetową. Jedną z dostępnych form prezentacji przebiegu choroby będą wykresy. Wykresy będzie można wydrukować lub wyeksportować do pliku PDF.

Aby skorzystać z aplikacji, pacjent będzie musiał utworzyć konto systemie. Oprócz tradycyjnej rejestracji, w systemie dostępna będzie możliwość powiązania istniejącego konta w sieci społecznościowej Facebook. Ta funkcja uprości i przyspieszy proces rejestracji.

1.1.2. Osoby bliskie

Osoba chora na cukrzycę będzie miała możliwość zdefiniowania grupy osób, które będą miały dostęp do danych pacjenta. Osoba, która otrzyma dostęp do przebiegu choroby będzie miała wgląd w przebieg choroby pacjenta. Dane będzie można obejrzeć również na wykresach. Taka osoba będzie otrzymywała na swój telefon powiadomienia o bieżących pomiarach cukru oraz przyjętej insulinie.

1.1.3. Lekarz

Lekarz podczas wizyty będzie mógł przejrzeć wykresy z przebiegu choroby z wybranego okresu czasu. Dostępne będą one w wersji elektronicznej oraz papierowej. Pomoże to w interpretacji danych i ustaleniu dalszego leczenia.

1.2. Przechowywanie danych w urządzeniu mobilnym

Do przechowywania danych na platformie mobilnej można wykorzystać lokalną bazę danych umieszczoną bezpośrednio w urządzeniu. Jest to bardzo dobre miejsce do przechowywania informacji zgromadzonych lub wygenerowanych przez aplikację, które są na sztywno powiązane z danym urządzeniem. Oznacza to, że w momencie odinstalowania takiego programu wszystkie zgromadzone w nim dane są tracone – baza danych jest usuwana wraz z aplikacją.

W aplikacji *Diabetes Diary* przechowywane będą dane między innymi o pomiarach glukozy i iniekcjach insuliny, które muszą zostać utrwalone w stałym magazynie danych. Dane powinny być przypisane do konta użytkownika a nie do konkretnej instancji aplikacji na konkretnym urządzeniu. Oznacza to, że użytkownik może zainstalować aplikację na wielu różnych urządzeniach i na każdym z nich będzie posiadał dostęp do tych samych, zawsze aktualnych danych. Dane pomiędzy tymi urządzeniami będą synchronizowane. Usunięcie aplikacji nie może spowodować usunięcia danych, które zgromadzi program.

Oprócz tego, aplikacja będzie wykorzystywała zmienne zbiory informacji, które co pewien czasu muszą być synchronizowane. Jednym z takich zbiorów będzie lista dostępnych na rynku insulin. Rynek insulin jest zmienny. Co pewien czas stare leki są zastępowane przez ich nowe odpowiedniki. Tego typu dane muszą być więc co jakiś czas synchronizowane, aby użytkownik zawsze posiadał dostęp do aktualnych danych.

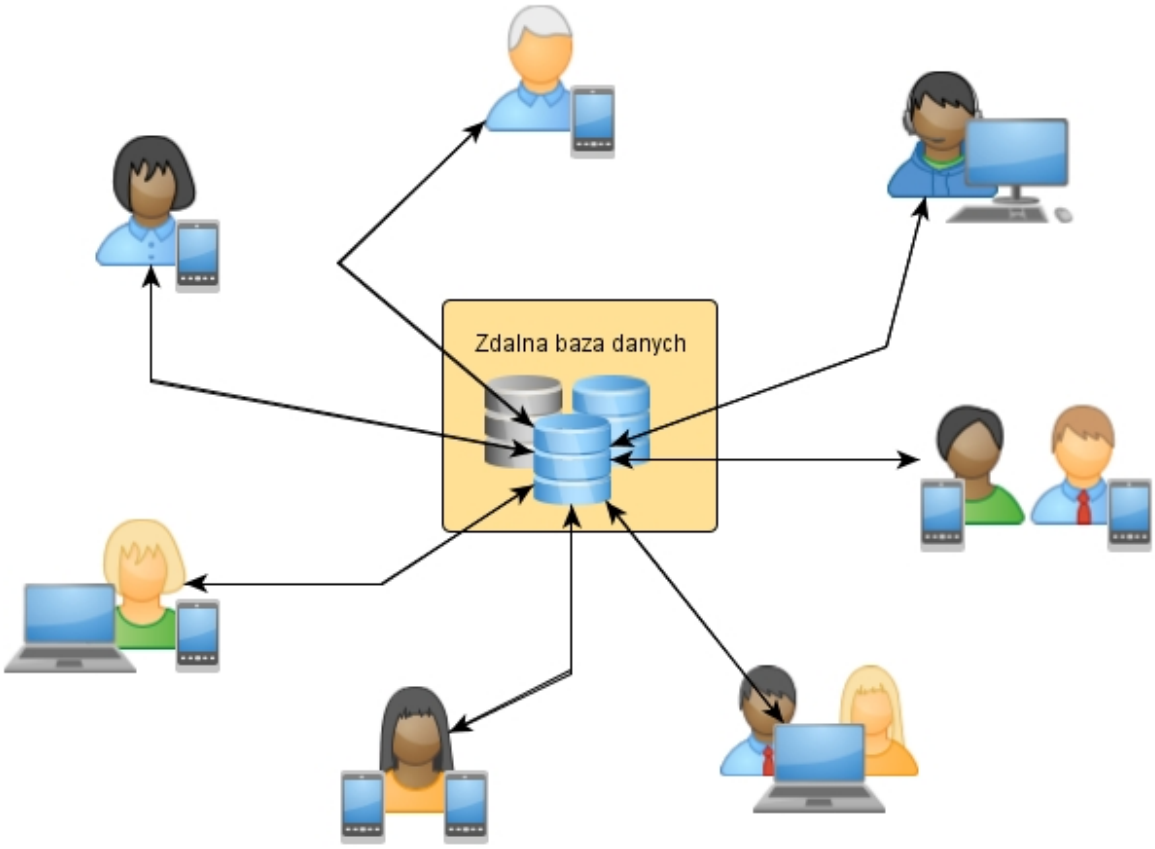
Powyższe przypadki znacząco ograniczają wykorzystanie lokalnej bazy danych jako główny magazyn informacji. Funkcja permanentnego utrwalania gromadzonych danych, oraz konieczność synchronizacji danych pomiędzy wieloma urządzeniami wymusza w takim przypadku skorzystanie z innego rozwiązania – zdalnej bazy danych dostępnej przez sieć Internet.

Takie zastosowanie nie eliminuje jednak całkowicie wykorzystania lokalnej bazy danych w aplikacjach rozproszonych. Lokalny magazyn informacji świetnie nadaje się jako cache danych oczekujących na synchronizację lub jako tymczasowe źródło danych w sytuacjach braku dostępu do internetu.

1.3. Zdalny magazyn danych

Problemy z synchronizacją danych pomiędzy wieloma różnymi urządzeniami można rozwiązać wykorzystując zdalny magazyn danych. W takim podejściu tworzony jest system rozproszony w którym wiele urządzeń wykorzystuje tę samą bazę danych, umieszczoną w zewnętrznym miejscu. Komunikacja z bazą danych odbywa się przez sieć internetową na zasadzie Klient – Serwer.

Serwer musi umożliwiać dostęp do bazy danych. Ze względów bezpieczeństwa dostęp do bazy powinien być ściśle kontrolowany. W tym celu należy utworzyć mechanizm sterujący dostępem do wybranych usług – tzw. API (ang. *Application Programming Interface*).



Ilustracja 2: System rozproszony - zdalna baza danych

W przeciwieństwie do lokalnej bazy danych z której w jednym czasie korzysta maksymalnie jedno urządzenie, zdalny magazyn informacji może być jednocześnie wykorzystywany przez bardzo wielu użytkowników. Serwer musi być więc jednostką na tyle wydajną, aby móc obsłużyć dużą ilość żądań.

2. Przegląd technologii

Na potrzeby niniejszej pracy powstaną trzy aplikacje wchodzące w skład całego systemu. Dwie z nich będą pracowały po stronie serwera, zaś trzecią z nich będzie aplikacja pracująca na platformie mobilnej. W skład pracy wchodzić będą:

- Aplikacja na telefon
- Aplikacja internetowa
- Zdalny magazyn danych wraz z interfejsem dostępowym

2.1. Bazy danych

Zasadniczą częścią projektu będzie centralna baza danych umieszczona na zewnętrznym serwerze. Wybór konkretnej bazy danych jest uwarunkowany jej przyszłym zastosowaniem, przewidywanym obciążeniem oraz budżetem. Należy więc uwzględnić następujące czynniki:

Struktura bazy danych

Aplikacja będzie gromadziła wiele różnych zbiorów danych. W szczególności będą to następujące dane:

- Dane użytkowników
- Dane dotyczące choroby, stosowanych insulin i glukometru
- Dostępne insuliny na rynku
- Pomiary stężenia glukozy
- Dane o iniekcjach insuliny
- Dane kontroli dostępu

Wszystkie struktury będą w jakiś sposób ze sobą powiązane. Na przykład, informacja o iniekcji insuliny będzie bezpośrednio powiązana z użytkownikiem, który dokonał wstrzyknięcia oraz z konkretnym rodzajem insuliny. Baza danych musi być więc oparta o ***model relacyjny***. Zbiory informacji będą ze sobą powiązane z pomocą relacji opisanych przez zestaw kluczy głównych oraz kluczy obcych. Takie podejście ułatwia wprowadzanie zmian i zmniejsza ryzyko pomyłek, ponieważ każda zmiana zbioru danych wiążę się z koniecznością sprawdzenia integralności bazy czyli sprawdzeniem poprawności relacji. Czuwanie nad integralnością odbywa się kosztem wydajności, jednak korzyści jakie z tego płyną są dużo większe niż straty osiągów.

Ilość danych

Ponieważ aplikacja będzie dostępna bez ograniczeń dla wszystkich użytkowników telefonów z całego świata, można spodziewać się, że w niedługim czasie w bazie danych pojawią się miliony danych. Baza danych musi być w stanie sprawnie

obsługiwać bardzo duże zbiory informacji.

Obciążenie

Zdalna baza danych umieszczona na zewnętrznym serwerze, będzie źródłem danych z którego korzystać będą wszystkie aplikacje uruchomione na platformie mobilnej. Ponieważ aplikacja będzie dostępna dla użytkowników na całym świecie, należy spodziewać się, że będzie ona pod bardzo dużym obciążeniem. Obciążenie może sięgać od kilkuset do kilku tysięcy zapytań na sekundę. Niezbędna więc będzie optymalizacja baza danych. Osiągnięte to będzie przez konstrukcję wydajnych zapytań na poziomie interfejsu dostępowego do bazy danych oraz poprzez nałożenie odpowiednich indeksów. Baza danych musi więc umożliwia tworzenie ***indeksów****.*

Wydajność

Baza danych będzie pracowała pod dużym obciążeniem. Ważne jest więc, aby sama baza była wydajna dla operacji wstawiania, wyszukiwania i łączenia wyników.

Budżet

Duży wpływ na wybór konkretnej bazy danych ma budżet. Na rynku istnieje wiele bardzo dobrych baz danych, takich jak chociażby *Oracle,* są to jednak rozwiązania komercyjne. Ponieważ projekt ma charakter niekomercyjny, poszukiwania zostaną ograniczone jedynie do wolnodostępnych baz danych.

Podsumowując powyższe rozważania, baza danych powinna być w stanie zgromadzić bardzo dużą ilość informacji zachowując przy tym wysoką wydajność. Powinna być też rozwiązaniem darmowym i ogólnodostępnym. Wybór ogranicza się więc do następujących systemów baz danych:

- **SQLite**
- **MySQL**
- **PostgreSQL**

2.1.1. Baza SQLite

Baza danych SQLite jest rozwiązaniem darmowym. Jej kod jest otwarty. Licencja

bazy umożliwia wykonywanie kopii, wprowadzanie własnych modyfikacji w kodzie źródłowym, wykorzystywanie zarówno w projektach indywidualnych jak i komercyjnych. Jej wykorzystanie jest wręcz nieograniczone przez autorów.

Zaletą tej bazy danych jest niewątpliwie jej prostota i nieduże wymagania. Została ona napisana w języku C. Jest ona w całości oparta na plikach, dlatego jedynymi wymaganymi do uruchomienia bibliotekami są te, które obsługują alokację, zapis i odczyt pamięci. Dzięki temu, baza danych może być bez problemów i ograniczeń wykorzystywana w ograniczonych urządzeniach, takich jak chociażby urządzenia mobilne.

W przeciwieństwie do innych rozwiązań, aby korzystać z bazy SQLite nie potrzebne jest uruchomienie dodatkowego procesu serwującego do niej dostęp. Proces, który chce pobrać lub zapisać dane w bazie, komunikuje się bezpośrednio z plikiem bazy danych umieszczonym na dysku. Zaletą tego rozwiązania jest brak konieczności instalacji i konfiguracji dodatkowego oprogramowania, które będzie sterowało dostępem do bazy danych. Praca z baza SQLite jest możliwa bez żadnej wcześniejszej konfiguracji, a każdy program jest w stanie zapisywać i odczytywać dane z wybranej bazy.

Pomimo prostoty, baza SQLite posiada dosyć duże możliwości. W aktualnej wersji SQLite3 do dyspozycji są następujące mechanizmy:

- **Klucze główne i klucze obce**
- **Indeksy**
- **Operacje na czasie i datach**
- **Obsługa podstawowych typów: NULL, INTEGER, REAL, TEXT, BLOB**
- **Transakcje**

Oprócz wymienionych zalet, baza SQLite posiada inne dosyć poważne problemy i ograniczenia. Głównym problemem jest brak kontroli dostępu do bazy danych. W SQLite nie istnieją mechanizmy, które sterują tym, kto może odczytywać i zapisywać informacje. Każdy proces, który zna lokalizację pliku z bazą danych może się z nią połączyć i wykonać potencjalnie niebezpieczne lub niepożądane operacje. Problem

ten można częściowo obejść wykorzystując inne mechanizmy systemu operacyjnego ograniczające dostęp do plików.

Kolejnym problemem z którym boryka się SQLite jest wielodostępowość. Proces, który odczytuje bądź zapisuje dane w bazie, nakłada na plik bazy blokadę, która jest zwalniana dopiero w momencie skończenia wykonywania operacji. Oznacza to, że w jednej chwili tylko jeden wątek może odczytywać bądź zapisywać dane. Pozostałe wątki są zatrzymywane i oczekują na zwolnienie blokady.

Ilość wad w porównaniu do ilości zalet SQLite jest niewiele. Są one jednak na tyle poważne, że jej wykorzystanie ogranicza się zazwyczaj jako lokalny magazyn danych. Problemy kontrolą dostępu oraz z wydajnością przy wielokrotnym dostępie wykluczają ją jako serwer danych.

2.1.2. Baza MySQL

MySQL jest jedną z najbardziej znanych i najczęściej wykorzystywanych baz danych. Jej licencja umożliwia darmowe wykorzystanie zarówno w projektach indywidualnych jak i w komercyjnych, pod warunkiem jednak, że kod aplikacji wykorzystującej bazę pozostanie otwarty. Początkowo podstawową zaletą MySQL była jej szybkość. Niestety oprócz dobrej wydajności, obsługi podstawowych typów i operacji, MySQL nie miał zbyt wiele do zaproponowania. Głównym zarzutem przeciwników bazy był brak obsługi transakcji. Obecna wersja MySQL 5.6 została uzupełniona o dużą ilość nowych, zaawansowanych mechanizmów przy zachowaniu bardzo dobrej wydajności. Najczęściej jest ona wykorzystywania do implementacji rozwiązań na potrzeby internetu w połączeniu z serwerem Apache'em i interpreterem języka skryptowego PHP.

Baza danych MySQL obsługuje kilka silników bazodanowych. Najczęściej wykorzystywanymi są silniki: *MyISAM* oraz *InnoDB*. Pierwszy z nich jeszcze do wersji 5.5 był silnikiem aktywnym domyślnie. Główną jego wadą jest brak obsługi transakcji oraz kluczy obcych. Drugi z nich wprowadza już pełną obsługę brakujących mechanizmów.

Typy danych

MySQL obsługuje podstawowe typy danych takie jak: INTEGER, TINYINT, MEDIUMINT, BIGINT, BOOLEAN, DECIMAL, FLOAT, DOUBLE, DATE, DATETIME, TIMESTAMP, CHAR, VARCHAR, TEXT oraz ENUM. Możliwe jest również przechowywanie danych w postaci binarnej w polu typu BLOB.

W obecnej wersji MySQL do dyspozycji są również takie mechanizmy jak:

- obsługa podzapytań
- klucze obce
- transakcje
- procedury składowane
- wyzwalacze
- widoki

2.1.3. Baza PostgreSQL [1]

Baza danych PostgreSQL jest uznawana za najbardziej zaawansowane darmowe narzędzie. Rozwijana jest od ponad 15 lat. Przez ten czas udało jej się zyskać dużą popularność i zaufanie. Kod bazy jest otwarty. Licencja umożliwia wykonywanie kopii, wprowadzanie własnych modyfikacji w kodzie źródłowym, wykorzystywanie zarówno w projektach indywidualnych jak i komercyjnych.

PostgreSQL posiada szeroki wachlarz możliwości i zaawansowanych mechanizmów:

Typy danych

Baza obsługuje podstawowe typy danych takie jak: INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, ENUM, DATE, INTERVAL oraz TIMESTAMP. Oprócz tego wbudowane są mechanizmy wspierające przechowywanie dużych plików binarnych takich jak zdjęcia, muzyka czy filmy. Do dyspozycji mamy również wiele innych typów danych takich jak:

- **Typy geometryczne**
    - **POINT** – pojedynczy punkt na płaszczyźnie
    - **LSEG** – segment linii

        - **BOX** – prostokąt
        - **PATH** – ścieżka, otwarta lub zamknięta lista połączonych punktów
        - **POLYGON** – podobne do PATH
        - **CIRCLE** – okrąg
    - **Adresy sieciowe**
        - **CIDR** – adresy sieci IPv4 oraz IPv6
        - **INET** – adresy sieci i hostów IPv4 oraz IPv5
        - **MACADDR** – numery kart sieciowych
    - **XML** – przechowywanie dokumentów XML
    - **JSON** – przechowywanie obiektów JSON (obiektów zapisanych w notacji Java Script)

PostrgeSQL wspiera złożone klucze główne, klucze obce z ograniczeniami oraz indeksy. W bazie dostępne są również operacje JOIN, podzapytania, widoki, wyzwalacze oraz procedury składowane. Wykorzystywać można również transakcje, których implementacja jest zgodna ze zbiorem zasad ACID *(ang. Atomicity Consitency Isolation Durability) [2].* Daje to gwarancję zachowania integralności na każdym etapie korzystania z bazy. Obsługiwane są również transakcje zagnieżdżone. Te wszystkie zaawansowane mechanizmy sprawiają, że PostgreSQL jest często wykorzystywany w dużych, zaawansowanych systemach.

Baza PostgreSQL jest rozwiązaniem szeroko skalowalnym. Dobrze daje sobie radę zarówno w w niewielkich systemach jak i w dużych komercyjnych projektach, gromadzących miliony informacji. Istnieje wiele aktywnych systemów w wersjach produkcyjnych, wykorzystujących PostgreSQL, których systemy zarządzają bazami o wielkościach przekraczających 4 terabajty. Oprócz tego, baza świetnie radzi sobie z jednoczesnym dostępem do danych przez wielu użytkowników, dlatego jest bardzo dobrym rozwiązaniem jako serwer danych.

Limit	Wartość
Maksymalny rozmiar bazy	Nieograniczony
Maksymalny rozmiar tabeli	32 TB
Maksymalny rozmiar krotki	1.6 TB
Maksymalny rozmiar pola	1 GB
Maksymalna liczba krotek	Nieograniczona
Maksymalna liczba kolumn w tabeli	Od 250 do 1600 w zależności od typów
Maksymalna liczba indeksów w tabeli	Nieograniczona

Tabela 1: Ograniczenia rozmiarowe i ilościowe bazy PostgreSQL

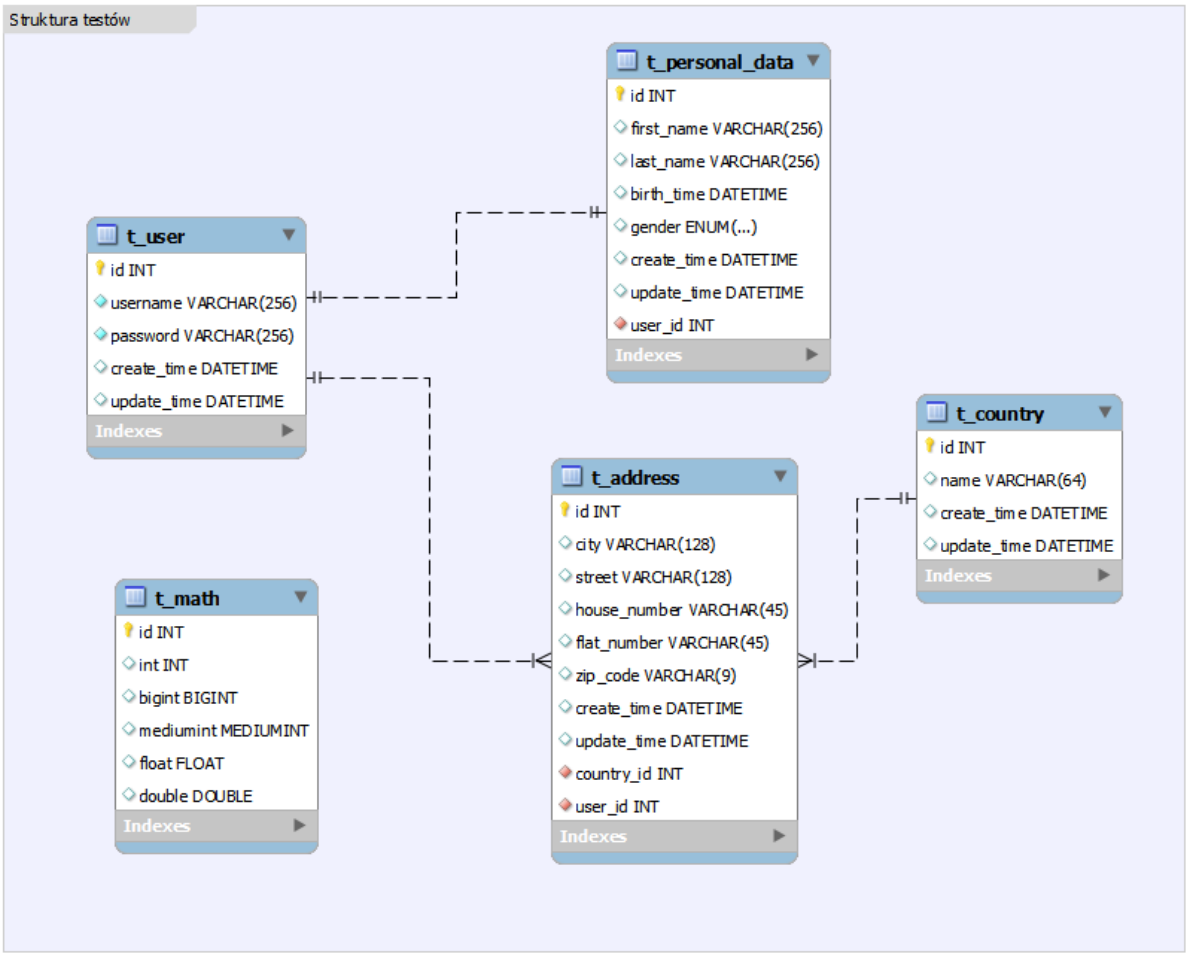
Najczęściej przytaczanym zarzutem wobec bazy PostgreSQL jest jej wydajność. Czas wykonywania operacji na bazie w wersjach równych bądź niższych niż 8.4.16 był relatywnie długi. Kolejne zapowiedziane przez PostgreSQL wersje oznaczone numerami 9.0 oraz 9.2 miały przynieść duży wzrost wydajności przy zachowaniu wszystkich dotychczasowych możliwości. Tak faktycznie się stało, nowa wersja przyniosła znaczne poprawienie osiągów – zarówno wydajności jak i skalowalności. Twórcy bazy przedstawiają poprawę osiągów następująco: [3]

- Do 350 000 odczytów na sekundę (ponad 4 razy szybciej)
- Do 14 000 zapisów danych na sekundę (5 razy szybciej)
- Do 30% mniejsze obciążenie procesorów

2.1.4. Test wydajności

Spośród wymienionych baz danych dwie z nich, MySQL oraz PostgreSQL, spełniają założenia projektu *Diabetes Diary* – są dobrym wyborem do implementacji zdalnego magazynu danych. Problemy z jednoczesnym dostępem wielu użytkowników do bazy SQLite eliminują ją jako dobre rozwiązanie dla serwera danych. Wybór pomiędzy MySQL a PostgreSQL nie jest jednak łatwy. Początkowo MySQL był zdecydowanie szybszy od PostgreSQL, zaś PostgreSQL był zdecydowanie bardziej zaawansowanym systemem niż MySQL. Wraz z rozwojem obu baz te różnice zostały zniwelowane – MySQL został wzbogacony o wiele zaawansowanych mechanizmów, zaś PostgreSQL został wyraźnie zoptymalizowany.

W podjęciu ostatecznej decyzji pomoże specjalnie przygotowany do tego celu zestaw testów wydajnościowych. Przetestowane zostaną wszystkie trzy bazy: SQLite, MySQL oraz PostgreSQL. Testy obejmować będą podstawowe operacje takie jak tworzenie tabel, wstawianie, usuwanie rekordów, złączenia tabel, operacje arytmetyczne, operacje na datach i ciągach znaków, podzapytania oraz wyszukiwanie po indeksach.



Ilustracja 3: Struktura tabel wykorzystana do testów wydajnościowych baz danych

Testy zostaną przeprowadzone na następujących bazach danych:

- SQLite w wersji 3
- MySQL w wersji 5.1 z silnikiem MyISAM
- MySQL w wersji 5.1 z silnikiem InnoDB
- PostgreSQL w wersji 8.3
- PostgreSQL w wersji 9.2

W każdej z tych baz utworzona zostanie taka sama struktura tabel. W testach udział wezmą następujące tabele:

- **t_user** – tabela przechowująca podstawowe dane użytkownika
- **t_personal_data** – tabela przechowująca szczegółowe informacje użytkownika, takie jak imię czy nazwisko. Tabela jest złączona kluczem obcym z tabelą *t_user*.

- **t_country** – tabela przechowująca listę krajów z całego świata.
- **t_address** – tabela przechowująca informacje o danych adresowych użytkowników. Występują w niej złączenia kluczami obcymi z tabelami *t_user* oraz *t_country*.
- **t_math** – tabela niezależna od pozostałych. Przechowuje ona typy na których przeprowadzony zostanie szereg operacji arytmetycznych.

Do przeprowadzenia testów utworzone zostało specjalne środowisko testowe, które wykona szereg identycznych testów na każdej z baz.

Testy zostały już przeprowadzone jednak ich wynik jest niejednoznaczny ponieważ zostały one wykonane na zewnętrznym serwerze o zmiennym obciążeniu, dlatego ich wyniki są niejednoznaczne. Testy zostaną przeprowadzone ponownie w neutralnych warunkach – wyniki zostaną zamieszczone i omówione w tym miejscu później.

2.2. Interfejs dostępowy do centralnego magazynu danych – API

Zdalna baza danych jest sercem systemu. Przechowywane w niej będą informacje zgromadzone przez wszystkich użytkowników korzystających z aplikacji mobilnej oraz internetowej. Będą się one komunikowały z bazą w celu pobrania i zapisu danych. W teorii taka komunikacja mogłaby odbywać się bezpośrednio poprzez nawiązanie połączenia ze zdalną bazą danych. Nie jest to jednak rozwiązanie bezpieczne. Aplikacje klienckie otrzymałyby bowiem swobodny dostęp do całego systemu, w tym potencjalnie wrażliwych danych. Najczęściej stosowanym rozwiązaniem tego problemu jest wykorzystanie wzorca projektowego *proxy* do implementacji pewnej „otoczki” sterującej dostępem do bazy danych – tzw. API *(ang. Application Programming Interface).* Interfejsem dostępowym będzie aplikacja pracująca po stronie serwera, jako jedyna bezpośrednio komunikująca się ze zdalna bazą danych. API będzie więc pośrednikiem w komunikacji pomiędzy aplikacją kliencką a serwerem danych. Sterować ono będzie dostępem do danych.

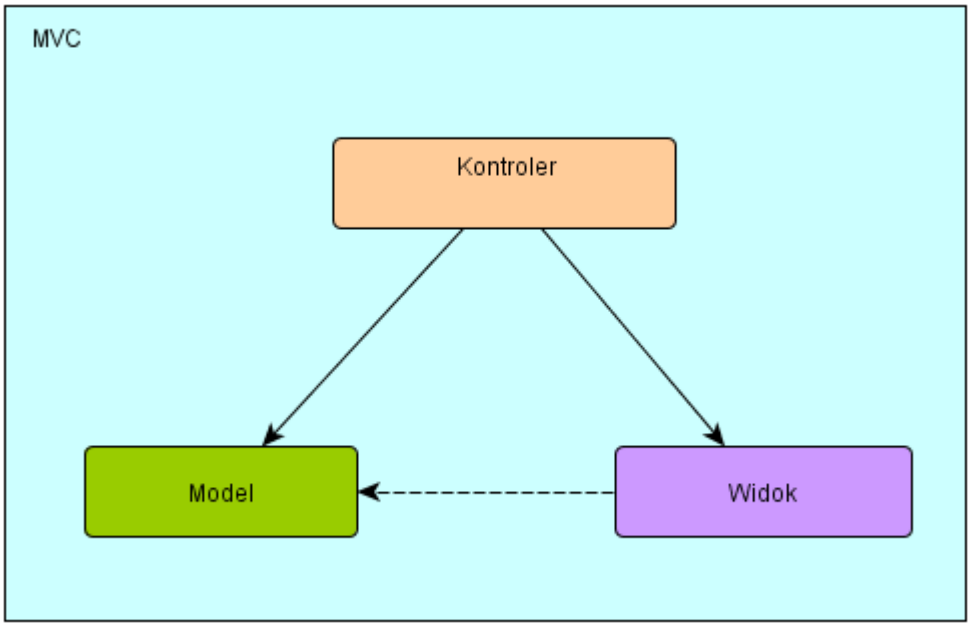
Ze względu na powszechną dostępność oraz niezawodność, interfejs zostanie zaimplementowany w języku skryptowym PHP z wykorzystaniem frameworka Yii.

2.2.1. Framework Yii

Framework Yii jest narzędziem wspomagającym tworzenie aplikacji w języku PHP. Pozwala ono zaoszczędzić programiście wiele godzin pracy, zwalniając z konieczności implementacji rozwiązań, które już wcześniej zostały znalezione i wypróbowane. Framework jest pewnego rodzaju rozszerzeniem standardowych możliwości języka, poprzez wprowadzenie zbioru nowych bibliotek, reguł i konwencji.

Wzorzec projektowy MVC

Yii jest frameworkiem opartym o wzorzec projektowy *MVC (ang. Model View Controller)*. Takie podejście odseparowuje od siebie etapy programowania, które zazwyczaj przenikały się i mieszały podczas programowania w czystym języku, wprowadzając nieład i komplikując kod programu.



Ilustracja 4: Warstwy wzorca projektowego MVC

Warstwa modelu – jest to podstawowa warstwa w której znajdują się definicje składowych systemu. Jest to więc zbiór klas opisujących obiekty pomiędzy którymi zachodzą interakcje.

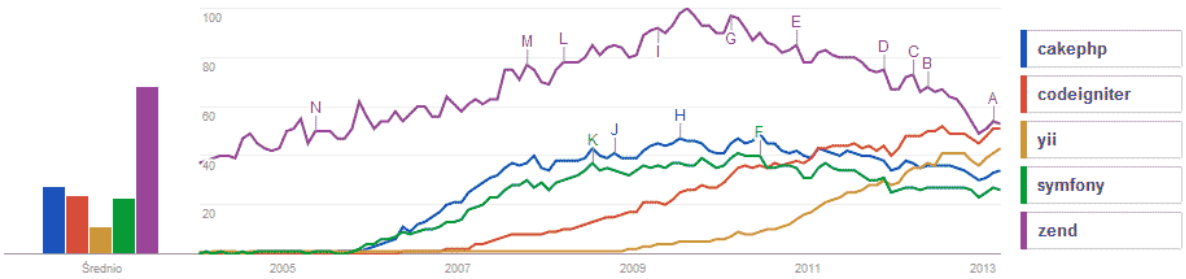
Warstwa kontrolera – jest to warstwa, która zajmuje się przeprowadzaniem konkretnych akcji systemu. Tutaj inicjowane są potrzebne do wykonania danej

czynności obiekty. Na utworzonych obiektach wykonywane są odpowiednie operacje. Na końcu, wyniki zwracane są do warstwy widoku, która te dane zaprezentuje.

Warstwa widoku – jest to warstwa, którą widzi użytkownik aplikacji. Wyświetlone są w niej dane przekazane przez kontroler. Dane zazwyczaj przekazane są w postaci modeli, które wykorzystywane są wyłącznie w trybie do odczytu.

Zalety i popularność Yii

Na rynku dostępnych jest co najmniej kilka dobrych frameworków. Do niedawna najbardziej popularnym ale również płatnym był *Zend Framework*. Jest on jednak uważany za dosyć ciężkie i powolne narzędzie. W przeciwieństwie do niego, Yii jest uznawane za „najlżejszy" i najszybszy framework PHP. Jest to ponadto rozwiązanie całkowicie bezpłatne, zarówno w projektach indywidualnych jak i komercyjnych. To sprawiło, że dzisiaj Yii jest trzecim pod względem popularności frameworkiem [4].



Ilustracja 5: Popularność wiodących frameworków PHP

Główne zalety Yii:

- **Znacząco przyspiesza pisanie aplikacji w PHP**
- **Bardzo szybki**
- **Oparty o wzorzec MVC**
- **Całkowicie darmowy**
- **Wsparcie dla relacyjnych baz danych**
- **Lazy-loading** – ładowanie potrzebnych danych dopiero w momencie kiedy są one potrzebne.
- **Eager-loading** – ładowanie od razu całego zestawu potrzebnych danych. Redukcja niepotrzebnych zapytań do bazy danych – zmniejszenie jej obciążenia.
- **Generatory modeli i kontrolerów** – framework potrafi wygenerować modele

klas na podstawie schematu tabel utworzonych w bazie danych.

- **Szeroka skalowalność**

2.2.2. Webservices

Dostęp do bazy danych będzie możliwy przez specjalnie do tego przygotowany zestaw metod interfejsu – tzw. *webservices*. Funkcje te sterować będą dostępem, dbając o to, aby operacje zapisu i odczytu danych mogły być wykonywane jedynie przez użytkowników zalogowanych i posiadających odpowiedni poziom uprawnień. Metody API wywoływane będą wykorzystując w tym celu protokół *HTTP*. Każda funkcja będzie posiadała swój unikalny adres pod który wysyłane będzie żądanie. Parametry metod przesyłane będą wraz z żądaniem. Podczas komunikacji z API przesyłane będą wrażliwe dane, które ze względów bezpieczeństwa nie mogą zostać przechwycone przez osoby niepożądane. Dane muszą być więc szyfrowane. W tym celu komunikacja z interfejsem dostępowym odbywać się będzie wykorzystując bezpieczne połączenie SSL.

2.2.3. Format wymiany danych

Metody API będą wykonywały różnego rodzaju operacje na bazie danych a następnie do klienta zwracane będą wyniki. Urządzenie docelowe musi być w stanie odczytać otrzymane informacje. W tym celu należy uzgodnić format wymiany danych za pomocą którego, serwer przekazywać będzie wyniki. W technologiach internetowych powszechnie stosowane są dwa formaty wymiany danych: *XML* oraz *JSON*.

Soap XML

Sekcja wymaga uzupełnienia

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="urn:ExchangeFormatTestControllerwsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <ns1:getAddressResponse>
         <return xsi:type="ns1:Address">
            <id xsi:type="xsd:integer">1</id>
            <city xsi:type="xsd:string">Lublin</city>
            <street xsi:nil="true" />
            <house_number xsi:type="xsd:string">123</house_number>
            <flat_number xsi:type="xsd:string">34</flat_number>
            <zip_code xsi:type="xsd:string">20-234</zip_code>
            <create_time xsi:type="xsd:string">2013-01-01 12:12:12</create_time>
            <update_time xsi:type="xsd:string">2013-01-01 12:12:12</update_time>
            <country_id xsi:type="xsd:integer">2</country_id>
            <user_id xsi:type="xsd:integer">3</user_id>
         </return>
      </ns1:getAddressResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Tabela 2: Przykład danych zapisanych w XML

JSON

Format JSON *(ang. JavaScript Object Notation)* był początkowo stosowany do serializacji i deserializacji obiektów w języku JavaScript. Notacja zapisu jest bardzo prosta i przejrzysta. Umożliwia ona serializację obiektów dowolnego typu – zarówno struktur prostych jak i wielokrotnie złożonych. Zalety tego formatu sprawiły, że obecnie jest on powszechnie wykorzystywany jako format wymiany danych pomiędzy usługami sieciowymi.

```
[
  {
    "id":"1",
    "city":"Lublin",
    "street":"Agatowa",
    "house_number":"123",
    "flat_number":"34",
    "zip_code":"20-234",
    "create_time":"2013-01-01 12:12:12",
    "update_time":"2013-01-01 12:12:12",
    "country_id":"2",
    "user_id":"3"
  }
]
```

Tabela 3: Przykład danych zapisanych w notacji JSON

W powyższych tabelach zwrócony został ten sam wynik, zapisany w obu formatach. Te same dane zapisane w XML są znacznie bardziej obszerne niż dane zapisane w notacji JSON. Zakładając, że system docelowo obsługiwać będzie tysiące użytkowników i tysiące żądań, należy mieć również na uwadze rozmiar przesyłanych danych. Dane zapisane w JSON będą przesłane do klienta znacznie szybciej, zużywając przy tym mniejszą ilość transferu.

Ze względu na prostotę implementacji, niski koszt wprowadzania zmian i szybkość parsowania, interfejs dostępowy do bazy danych systemu *Diabetes Diary* do komunikacji będzie wykorzystywał format *JSON*.

2.3. Aplikacja mobilna

Aplikacja mobilna zostanie napisana na platformę mobilną Android. Docelowo program będzie dostępny na telefony i tablety pracujące pod aktualna wersją systemu oznaczoną numerem 4.0 lub wyższą.

Ta sekcja wymaga uzupełnienia

2.4. Aplikacja internetowe

Stworzony zostanie również panel administracyjny systemu w którym administrator będzie miał wgląd do wszystkich danych zgromadzonych w systemie.

Ta sekcja wymaga uzupełnienia

3. Projekt i implementacja centralnego magazynu danych z interfejsem dostępowym – API

3.1. Przegląd potrzebnych metod

3.2. Projekt bazy danych

3.3. Mechanizm bezpiecznego uwierzytelniania użytkowników

3.4. Format wymiany danych

3.5. Obsługa błędów

3.6. Implementacja metod

Bibliografia

1: , , 2013, http://www.postgresql.org/about/

2: , , 2012, http://pl.wikipedia.org/wiki/ACID

3: , , 2012, http://www.postgresql.org/about/press/presskit92/pl/

4: , , 2013, http://www.google.com/trends/explore#q=cakephp,codeigniter,yii,symfony