POLITECHNIKA WARSZAWSKA

Wydział Elektroniki i Technik Informacyjnych

Instytut Telekomunikacji

**PRACA DYPLOMOWA**

**INŻYNIERSKA**

**Sebastian Łuczak, Maciej Nowak**

**TYTUŁ**

Kierujący pracą:

mgr inż. Marcin Golański

..........................................

ocena pracy

..........................................

data i podpis Przewodniczącego

Komisji Egzaminacyjnej

Warszawa, wrzesień 2011

## SPIS TREŚCI :

1. Wstęp
2. Cel i zakres pracy
3. wstęp teoretyczny
   1. opis technologii
      1. ~~android~~
      2. ~~nexus~~
      3. ~~oracle~~
      4. nfc
      5. ~~Mifate Ultralight~~
      6. ~~J2EE~~
      7. ~~Spring~~
      8. ~~Hibernate~~
      9. ~~Jersey~~
      10. ~~Restful application – json~~
      11. eclipse
   2. przyczyny doboru ?
4. ~~koncepcja pracy~~
5. opis systemu
   1. ~~scenariusz użycia~~
   2. opis telefonu
      1. ~~aplikacja rejestrująca~~
      2. aplikacja administratora
   3. ~~opis Web-service – usługa internetowa~~
      1. ~~4 pod serwisy~~
   4. ~~opis bazy danych~~
   5. opis panelu administracyjnego
6. korzyści płynące z rozwiązania
7. podsumowanie
8. wnioski
9. bibliografia

# Wstęp

# Cel i zakres pracy

Celem naszej pracy inżynierskiej jest zrealizowanie kompletnego i nowatorskiego systemu kontroli czasu pracy z przeznaczeniem dla pracowników mobilnych. Innowacyjne wykorzystanie wchodzącej na rynek technologii NFC (Near Field Communication) , telefonii komórkowej oraz usług internetowych czyni nasze rozwiązanie unikalnym na rynku.

Dzięki infrastrukturze telefonii 3G, oraz złożonemu systemowi monitorowania uzyskujemy pełną i pewną kontrolę nad czasem poświęconym przez pracownika na realizację powierzonych mu zadań.

Kompletna realizacja zakłada dostarczenie zestawu aplikacji współpracujących ze sobą, na który składają się: baza danych, aplikacja serwerowa będąca usługą sieciową, internetowy portal informacyjny o przeznaczeniu administracyjnym oraz dwie łączące się z nimi aplikacje mobilne (kliencka i administracyjna) na telefon komórkowy z systemem Android. Każdy z tych elementów zostanie przedstawiony oddzielnie w dalszej części pracy.

Technologie użyte do wykonania pracy są powszechnie wykorzystywane do wytwarzania oprogramowania biznesowego przeznaczonego dla dużej ilości jednoczesnych użytkowników i zgodne z powszechnymi trendami realizacji usług w oparciu o sieć Internet.

Prezentowana realizacja ma być w założeniu alternatywą dla istniejących już systemów opartych o architektury stacjonarne, zrealizowane przy pomocy stacjonarnych czytników RFiD i kart zbliżeniowych, a także rozwiązaniem problemu kontroli czasu pracy osób których profil zatrudnienia zakłada stałe przemieszczanie się po kraju w celu serwisowania urządzeń bezpośrednio u klienta.

Dotychczasowe systemy ze względu na złożoną, specyficzną i kosztowną infrastrukturę nie odpowiadały na potrzeby dużej grupy pracodawców, którzy współpracę z kontrahentami opierają na umowach okresowych (serwis urządzeń, wsparcie techniczne IT itp.), a wynagrodzenie pracowników uzależniają od czasu poświęconego na wykonanie zlecenia. W takiej sytuacji potrzebna jest duża łatwość modyfikacji rozmieszczenia punktów kontrolnych, oraz ich bezproblemowa instalacja u kontrahentów, nie wymagająca żadnych narzędzi i ingerencji w otoczenie, a także zapewnienie transparentności i łatwości użytkowania dla pracowników.

Zarówno koncepcja jak i projekt aplikacji został wymyślony i zrealizowany wspólnie. Ze względu na modułowość zaprojektowanego przez nas rozwiązania, dokonaliśmy podziału pracy nad implementacją systemu na dwie części – mobilną i stacjonarną, leżącą po stronie serwera.

Aplikacje przeznaczone na telefon komórkowy stworzył Sebastian Łuczak, z kolei aplikacje usługi internetowej oraz portalu administracyjnego Maciej Nowak.

W dalszej części pracy przedstawione zostaną nasze rozwiązania na temat scenariuszu użycia kompletnego systemu, przyczyny doboru poszczególnych technologii oraz powody zastosowania konkretnych technik implementacji.

# Wstęp teoretyczny

Rozpoczynając projekt spędziliśmy kilka miesięcy na porównywaniu przeróżnych technologii i doborze optymalnych do realizacji naszego systemu. Pierwsza faza obfitowała w drobne prototypy dzięki którym wyeliminowaliśmy błędne założenia i elementy nadmiarowe, które nie dawały oczekiwanych rezultatów, przy jednoczesnym pogarszaniu jakości efektu końcowego.

Pierwsze tygodnie implementacji stanowiły dowód, że gruntowne przygotowanie podłoża, ustalenie zbioru potrzebnych aplikacji i faza prototypowania były warte poświęconego czasu, gdyż dzięki temu, dysponując wszystkimi niezbędnymi narzędziami, mogliśmy zabrać się do konsekwentnego rozwoju projektu.

## Opis technologii

### NFC

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

BLALBLALBALBLABLABLABLABLALB

## Android

Jednym z kluczowych czynników doboru technologii w której zrealizowana miała być część pracy zwana przez nas mobilną, była popularność danego rozwiązania na rynku konsumenckim tak, aby uzyskany system miał również zastosowanie pozanaukowe. Po zapoznaniu się z obecnym podziałem rynku okazało się, że 43,4% (Dane z Sierpnia 2011, Gartner) obecnie sprzedawanych urządzeń typu smartphone, wyposażona jest w system Android. Dodatkowo, po obserwacji dostępności terminali wyposażonych w system NFC, jasnym stało się, że również w tym wypadku system ten wiedzie prym, zarówno pod względem jakości dostarczonego API i uniwersalności (DOWODY), jak i liczby zapowiedzianych modeli wyposażonych w tę technologię.

Konsekwencją popularności Androida jest duży dostęp do różnorodnych opracowań tematu rozwoju i pielęgnacji aplikacji pisanych z przeznaczeniem na ten system i ogromne wsparcie ze strony użytkowników oraz samych twórców systemu, co zaowocowało dostępnością wielu bibliotek rozszerzających możliwości programistyczne.

Jako otwarta platforma Android daje programiście dostęp do obsługi praktycznie wszystkich podzespołów urządzenia i magazynów danych, z drugiej strony ścisłe wersjonowanie daje gwarancję, że aplikacja napisana z przeznaczeniem na dane wydanie API, będzie działała tak samo na różnych modelach urządzeń przenośnych, spełniających wymogi zdefiniowane w pliku AndroidManifest.xml danej aplikacji.

W pierwszej fazie projektowania aplikacji zmuszeni byliśmy do uwzględnienia faktu iż Android Gingerbread w wersji 2.3 nie dostarczał interfejsu programistycznego pozwalającego na pełną obsługę NFC. Dopiero aktualizacje do systemu operacyjnego zniosły z telefonów sztuczne ograniczenia. W pierwszej wersji Gingerbread niemożliwe między innymi było zapisywanie informacji na znacznik NFC (tylko odczyt), obsługiwana była mniejsza liczba dostępnych typów znaczników i nie istniała możliwość obsłużenia zbliżonego do urządzenia znacznika przez aplikację obecnie uruchomioną na pierwszym planie (system zawsze pytał się która aplikacja ma być wywołana dla danego znacznika).(DOWÓD)

Początkowy projekt aplikacji zakładał występowanie tych ograniczeń, szybko jednak okazało się, że w aktualizacji 2.3.3 (DATA) wspomniane funkcje zostały dostarczone, a dostępne wcześniej API, zmodyfikowane i w wielu przypadkach zauważalnie poprawione (DOWÓD).

Jak jednak uczy doświadczenie, każde rozszerzenie może przynieść rezultat odwrotny do oczekiwanego, dlatego też poddaliśmy pod ostrożną rozwagę poszerzenie funkcji realizowanych przez terminal mobilny.

Po pierwszych prototypach wykorzystujących nowe możliwości, okazało się, że część zaproponowanych usprawnień może faktycznie pozytywnie wpłynąć na komfort użytkowania końcowej aplikacji, przyspieszając jej wywołanie i zapewniając reakcję tylko na znaczniki określonego typu. Chodzi tu o bibliotekę **nfc.tech**, w oparciu o którą system dokonuje wstępnego doboru puli aplikacji zdolnych do obsłużenia danego znacznika. Dzięki temu nasza aplikacja reaguje tylko na zbliżenie do modelu znacznika Mifare Ultralight, a gdy jest aplikacją obecnie uruchomioną (pierwszego planu), przejmuje jego obsługę automatycznie, bez ingerencji ze strony użytkownika.

Metoda enableForegroundDispatch() która odpowiedzialna jest za przekazywanie intencji zawierającej w sobie dane odczytanego znacznika do obecnie wywołanej aplikacji nie jest jednak pozbawiona wad. W trakcie testów okazało się, że w około **10%** przypadków nie zachowuje się ona zgodnie z założeniem i system pyta się użytkownika która aplikacja ma zostać wywołana, choć powinien przekazać dane do aplikacji pierwszego planu.

Korzystając z systemu Andoid monitorowaliśmy również zużycie baterii przez moduł komunikacji NFC, jego poziom utrzymuje się poniżej jednego procenta całej konsumpcji.

**Google Samsung NEXUS S**

Ponieważ pracownicy mobilni często wyposażeni są w urządzenia typu smartphone zapewnione przez pracodawców, stwierdziliśmy, że docelową platformą musi być właśnie rozwiązanie tego typu. W konsekwencji wyboru systemu operacyjnego Android i wymogu posiadania modułu NFC, dobór urządzenia na którym powstać miała aplikacja był już w istotnym zakresie zdeterminowany. Krótkie rozeznanie rynku zakończyło się stwierdzeniem faktu – jedynym ówcześnie dostępnym telefonem spełniającym te wymagania jest Google Samsung Nexus S. Po oficjalnej aktualizacji oprogramowania do wersji Android 2.3.4, telefon spełnił wszystkie złożone w nim oczekiwania.

Znacznik NFC Forum Type 2 - Mifare Ultralight

Jedną z kluczowych zalet prezentowanego systemu jest jego cena wdrożeniowa i eksploatacyjna. Między innymi z tego powodu zdecydowaliśmy się na wykorzystanie niezwykle tanich i prostych konstrukcyjnie znaczników NFC Forum Type 2 Mifare Ultralight w formie miękkiej nalepki, w której cały układ wprasowany jest w papier.

Znaczniki te cechują się pojemnością zaledwie 384 bitów dostępnych na dane użytkownika w trybie zapis/odczyt, wyposażone są w system unikania kolizji, oferują prędkość transmisji 106 kbit/s, a także dostarczają możliwość jednokrotnego i nieodwracalnego zablokowania zapisu na znaczniku. Wadą tego rozwiązania jest jednak niemożność zmiany kodów dostępu do odczytu, przez co wszystko co jest zapisane na znaczniku, może być odczytane przez urządzenie zgodne ze standardem ISO/IEC 14443 **(INNOVA + MIFARE DATASHEET).**

Cechą która stała się krytyczna dla naszego systemu okazał się fakt iż każdy znacznik posiada zapisany na 7 bajtach unikatowy numer seryjny, zwany dalej identyfikatorem.

Ostatecznie postanowiliśmy, że na znaczniku zapisywana będzie tylko nazwa systemu, a jedyną informacją odczytywaną ze znacznika będzie jego identyfikator, który w bazie danych powiązany będzie z kompletem potrzebnych informacji. Dokładny opis wspomnianych danych zawarty jest w rozdziale opisującym projekt bazy danych. **(Dodatku A?)**

## Hibernate 3 :

Framework Hibernate w wersji 3 jest najbardziej popularnym rozwiązaniem programistycznym oferowanym na zasadach open-source, które realizuje podejście mapowania obiektów na struktury danych składowanych w relacyjnych bazach danych (ang. ORM = Object-to-Relational Mapping). Hibernate odpowiedzialny jest za mapowanie danych w postaci obiektów POJO (ang. Plain Old Java Object) do postaci rekordów w bazie danych jak i odwrotnie. Umożliwia on zapisywanie oraz pobieranie obiektów Java bez konieczności pisania zapytań w języku SQL.

Dzięki niewielkiemu nakładowi pracy potrzebnemu na konfigurację, możliwą zarówno za pośrednictwem plików xml jak i adnotacji zamieszczonych bezpośrednio w kodzie aplikacji programista skupia się na logice wytwarzanego oprogramowania podczas gdy Hibernate dba o zarządzanie sesją, transakcjami oraz o poprawne przechowywaniu danych z obiektów w encjach bazy danych.

Ze względu na różnicę pomiędzy bazami danych Hibernate wprowadza mechanizm dialektów który je usuwa oraz ukrywa przed programistą za interfejsem Hibernate.

Hibernate udostępnia programistom bardzo szerokie API ułatwiające tworzenie zapytań, konstruowanie filtrów a także stronicowanie wyników pobieranych z bazy danych. Dzieki takim udogodnieniom jak HQL (ang. Hibernate Query Language), QBE API (ang. Query By Example) czy też Criteria API operacje na rekordach bazy danych stają się o wiele łatwiejsze w porównaniu do wykorzystywania standardowego połączenia z bazą danych JDBC (ang. Java DataBase Connectivity).

W stworzonym przez nas systemie pełni on rolę głównego ogniwa łączącego warstwy logiki biznesowej z warstwą modelu danych. Wykorzystywany jest zarówno w aplikacji usługi sieciowej jak i aplikacji panelu administracyjnego.

## Spring Framework 3

Kolejnym Frameworkiem wykorzystywanym w naszej pracy jest Spring w wersji 3.0.1. Jest to środowisko które łączy ze sobą w harmonijną całość liczne interfejsy API.

Dzięki mechanizmowi wstrzykiwania zależności (ang. dependency injection ) udostępnia on cały szereg klas opakowujących popularne biblioteki Javy przez co ich użycie staje się znaczenie prostsze.

Oferuje on wsparcie dla wszystkich elementów potrzebnych do stworzenia zarówno aplikacji desktopowej jak i aplikacji internetowej. Implementuje sprawdzone wzorce projektowe a dzięki doskonale stworzonej dokumentacji ułatwia i promuje dobre praktyki programistyczne.

Posiada on również bardzo rozwinięte klasy wspierające integrację z innymi szkieletami aplikacyjnym takimi jak np. Hibernate czy też Log4j.

Spring składa się z wielu niezależnych od siebie modułów między innymi z pakietu Spring MVC oraz Spring Security które wykorzystujemy w naszym systemie.

Spring MVC (ang. Model – View – Controller ) jest to implementacja wzorca projektowego Model-Widok-Kontroler. Zakłada on rozdzielenie warstwy modelu – danych oraz logiki biznesowej od warstwy Widoku najczęściej stron JSP pokazujących dane. Przepływem pomiędzy tymi warstwami zarządza kontroler który także otrzymuje dane wejściowe od użytkowników i zarządza generowaniem odpowiednich widoków.

Spring Security to kolejny pakiet projektu Spring używany w naszym systemie. Do jego zadania należy zapewnienie bezpieczeństwa panelu administracyjnego. Dzięki jego udogodnieniom w łatwy sposób możemy kontrolować dostęp użytkowników do konkretnych funkcjonalności portalu. Moduł ten jest odpowiedzialny za uwierzytelnianie użytkowników a więc potwierdzenie ich tożsamości oraz za autoryzację czyli udzielenie bądź też pozbawienie dostępu do określonych części systemu.

Dzięki swojemu zaawansowaniu wspiera on wiele metod uwierzytelnienia takich jak na przykład LDAP czy też protokół Kerberos. W naszym przypadku uwierzytelnianie użytkownika opiera się na sprawdzeniu jego danych z tymi zapisanymi w bazie danych.

## Oracle 10g Express Edition

W naszej pracy zdecydowaliśmy się na wykorzystanie systemu zarządzania bazą danych Oracle w wersji 10g Express Edition stworzonego przez firmę Oracle Corporation. Edycja XE jest to darmowa wersja serwera bazodanowego Oracle 10g lecz ograniczona funkcjonalnie. Nie jest ona w stanie obsłużyć więcej niż jednego procesora oraz więcej niż 4GB pamięci RAM.

Pomimo swoich ograniczeń serwer zachowuje swoją wydajność oraz niezawodność. Dzięki rozbudowanym możliwością administracji oraz programom pokrewnym takim jak Oracle SQL Developer czy też Oracle Database Modeler mieliśmy możliwość łatwego zbudowania schematu bazy danych a także jego późniejszego zaimplementowania.

Wybraliśmy bazę danych firmy Oracle ze względu na chęć stworzenia systemu opartego na profesjonalnych komponentach składowych. Dzięki swojej popularności nie ma żadnych problemów w integracji bazy danych z frameworkiem Hibernate.

## Java 2 Enterprise Edition

Java jest to obiektowy język programownia udostępniony przez firmę Sun Microsystems w 1995 roku. Dziki swojej prostocie a jednocześnie zaawansowanym możliwościom szybko zyskała miano jednego z najpopularniejszych języków programowania na świecie.

Największą zaletą języka Java jest jego niezależność od platformy sprzętowej. Programy stworzone w tym języku można bez problemu uruchomić na wszystkich systemach operacyjnych np. Windows, Linux, Solaris. Jest to możliwe dzięki stworzeniu środowiska uruchomieniowego JRE (ang. Java Runtime Enviroment ), które składa się z wirtualnej maszyny Java JVM (ang. Java Virtual Machine ).

Java 2 EE jest to szeroko rozpowszechniona oraz używana platforma programistyczna opracowaną dla języka Java. Definiuje ona standardy oraz wzorce projektowe dla tworzenia oprogramowania opartego o wielowarstwową architekturę komponentową uruchamianego na serwerach aplikacyjnych. W jej skład wchodzą serwety, komponenty EJB (ang. Enterprise JavaBeans), usługi Web Service, pliki JSP, XML oraz inne elementy ściśle związane z wytwarzaniem aplikacji WWW takie jak HTML, CSS.

Standard J2EE obejmuje liczne interfejsy wspomagające programistę np. przy zapewnianiu bezpieczeństwa, definiowaniu interfejsu użytkownika czy też wysyłaniu poczty elektronicznej.

## Apache Tomcat

Nierozłącznym elementem aplikacji WWW jest serwer. W naszym projekcie wykorzystujemy darmową, rozpowszechnianą na zasadzie open-source implementacje Apache Tomcat 6.0.32. Jest to kontener aplikacji webowych będący serwerem, który umożliwia uruchamianie aplikacji internetowych wykonanych w technologiach Java servlets lub też JSP (ang. Java Server Pages). dopisać

## Architektura REST

REST ( ang. Representational State Transfer ) jest to wzorzec architektoniczny oparty na standardzie aplikacji internetowych tworzonych według modelu klient-serwer komunikujących się poprzez protokół HTTP. Wprowadza on szereg usprawnień i udogodnień mających na celu poprawienie wydajności, skalowalności oraz edycyjności usług internetowych (ang. Web Services).

Podstawowym założeniem wspomnianej architektury jest traktowanie danych oraz funkcjonalności jako zasobów łatwo dostępnych poprzez identyfikatory URI (ang Uniform Resources Identifiers), będących najczęściej hiperłączami w sieci. Dzięki mechanizmom zawartym w implementacji jest możliwe przesyłanie zasobów posiadających wiele różnych reprezentacji np. text, xml, json.

Jest on alternatywą dla dobrze znanego standardu SOAP (ang. Simple Object Access Protocol ) oferującego cały wachlarz zaawansowanych możliwości, a także będącego neutralnym dla protokołów warstwy transportowej. Niestety jego główną wadą jest potrzeba złożonej konfiguracji oraz opisu

Udogodnienia te sprawiają, że aplikacje oparte na tym wzorcu stają się proste, lekkie, oraz bardzo wydajne.

Architektura ta posiada wiele implementacji, jedną z nich jest Projekt Jersey, wykorzystywany w naszej pracy do stworzenia usługi internetowej komunikującej się z telefonem komórkowym.

Jersey jest wolno dostępną implementacją JAX-RS (ang. Java API for RESTful Web Services) udostępniającą mechanizm adnotacji, który to jeszcze bardziej ułatwia programistom tworzenie aplikacji.

# JSON - JavaScript Object Notation

JSON jest to prosty format danych, możliwy do odczytania przez człowieka, służący do wymiany informacji. Początkowo został on stworzony dla języka JavaScript lecz szybko stał się niezależnym formatem tekstowym posiadającym implementacje w wielu językach programowania między innymi Java. Jego właściwości sprawiają, że staje się on idealnym narzędziem do transmisji uporządkowanych danych poprzez sieć.

Jest on alternatywą dla powszechnie wykorzystywanego w Internecie języka XML (ang. eXtensible Markup Language ), który jednak wymaga większych nakładów programistycznych – dodatkowe klasy i obiekty – przy przetwarzaniu danych.

Poniżej prezentuję przykładowy format danych przesyłany w naszym systemie pomiędzy telefonem komórkowym a usługą sieciową.

Klasa CResponse zapisana w notacji JSON:

{

„STATUS” : „EVENT\_STATUS\_CREATED”,

„MESSAGE” : „przykładowa wiadomość”,

„TOKEN”: „2cabeae9a6c1febc74f2f8d9af6391104952445f”

}

.

# Scenariusz użycia:

Aplikacja może znaleźć zastosowanie w firmach zajmujących się serwisowaniem urządzeń położonych w różnych zewnętrznych lokalizacjach należących do zewnętrznych kontrahentów. Przypuśćmy taką sytuację. Nasza firma zajmuję się serwisowaniem urządzeń wielofunkcyjnych – kopiarek, skanerów, drukarek.

Pracodawca chcąc wdrożyć system, instaluje serwer usługi internetowej i portalu informacyjnego i wprowadza dane pracowników do bazy. Do pracowników przypisuje urządzenia mobilne definiując numery IMSI i IMEI. Każdy pracownik posiada również identyfikator i hasło do systemu.

Pewna firma podpisuje z nami umowę na usługę serwisowania urządzeń biurowych na miejscu, w jej siedzibie.

W momencie podpisania umowy z kontrahentem, przy pomocy mobilnej aplikacji administracyjnej do systemu wprowadzane są podstawowe dane kontrahenta i urządzenia na obudowie którego umieszczana jest naklejka ze znacznikiem RFiD (NFC) i ewentualnym logiem naszej firmy. W tym samym procesie unikatowy identyfikator znacznika wiązany jest w systemie z konkretnym urządzeniem.

W sytuacji w której kontrahent zwraca się z potrzebą wykonania usługi serwisowej, osoba przyjmująca zlecenie (operator) wprowadza dane do systemu i tworzy zdarzenie oczekujące na reakcję pracownika mobilnego. Pracownik mobilny otrzymuje powiadomienie w postaci wiadomości e-mail.

Na miejscu realizacji zlecenia, pracownik mobilny zbliża telefon do znacznika naklejonego na obudowie obecnie serwisowanego urządzenia. Aplikacja zainstalowana na aparacie automatycznie odbiera dane ze znacznika i kontaktuje się z serwerem w celu zgłoszenia rozpoczęcia realizacji usługi. Serwer weryfikuje tożsamość użytkownika i rejestruje godzinę rozpoczęcia obsługi zdarzenia. Aplikacja na telefonie potwierdza zarejestrowanie danych na serwerze i wyświetla informacje dodatkowe wprowadzone przez operatora.

Po zakończeniu usługi, pracownik mobilny ponownie przybliża telefon do znacznika i potwierdza chęć zakończenia realizacji zlecenia. Serwer potwierdza zakończenie zlecenia, odnotowuje moment końca pracy i zmienia stan zlecenia na zakończone.

Telefon sygnalizuje pomyślne zakończenie pracy i przełącza się w tryb oczekiwania na następne działanie.

Istotnym jest zaznaczenie tego, że w jednym momencie dany pracownik może realizować tylko jedno zgłoszenie (co również jest przekazywane w postaci czytelnej informacji do pracownika podczas próby rozpoczęcia kolejnego zlecenia).

Pracodawca lub operator mogą sprawdzić w panelu administracyjnym stany realizacji wszystkich zleceń oraz czas jaki został poświęcony na realizację konkretnej czynności.

# Koncepcja systemu

## Wstęp

W poniższym rozdziale postaramy się przybliżyć główne założenia oraz budowę wymyślonego i zaprojektowanego przez nas systemu. System ten ma za zadanie ułatwić nadzór nad pracownikami mobilnymi jednocześnie umożliwiając rzetelne rozliczenie ich czasu pracy. W naszym rozwiązaniu dokonaliśmy odwrócenia klasycznego scenariusza w którym czytnik RFiD ulokowany jest na stałe w ścianie lub bramce, a użytkownik posiada kartę RFiD. Sytuacja ta ma ekonomiczne uzasadnienie w sytuacji w której mówimy o stałej lokalizacji pracowników i ich dużej liczbie. Problemy pojawiają się jednak gdy lokalizacji jest wiele (tyle ilu klientów) i mogą się zmieniać, pracownicy stale poruszają się pomiędzy nimi, a pracodawcy zależy na informacjach na temat jakości obsługi klienta i sumienności swoich pracowników. W takiej sytuacji drogi czytnik musiałby zostać umieszczony w dziesiątkach jak nie setkach punktów, a całość musiałaby być spięta w dodatkową sieć zarządzającą dostępem.

W sytuacji w której czytnikiem jest telefon, a zatem urządzenie w które z powodów cywilizacyjnych każdy pracownik tego typu jest wyposażony, nie ma problemu związanego z dodatkowym kosztem (gdyż zawiera się w cenie urządzenia domyślnie używanego przez pracownika), a niezwykle (CENA!) tanie Tagi RFiD zgodne ze standardem NFC rozmieszczane zostają na każdym z serwisowanych urządzeń, powodujemy zmianę rozkładu kosztów i zwiększenie szczegółowości informacji (gdyż wiemy dokładnie nad którym zleceniem pracuje w danym momencie dany pracownik).

Dane transmitowane są w czasie rzeczywistym, dzięki czemu koordynator zna obecny stan zajętości zasobów ludzkich, a także jest w stanie ustalić z większą dokładnością średni czas realizacji danej usługi serwisowej.

Oczywiście zauważalną wadą zaproponowanego rozwiązania jest wrażliwość na dostępność do usług transmisji danych, niestety jednak wprowadzenie opcji pracy w trybie offline mogłoby spowodować ogromną wrażliwość na przekłamanie danych docierających do bazy, a raczej celową modyfikację daty systemowej przez użytkowników. Co prawda baza przechowuje dla każdego wpisu zarówno znacznik czasowy terminala użytkownika jak i serwera obsługującego zdarzenie, jednak zakładając działanie w trybie offline i kolejkowanie wysyłania danych do serwera, użytkownik mógłby celowo emulować przebywanie poza zasięgiem sieci tak, aby zamaskować opóźnione rozpoczęcie lub zakończenie realizacji zlecenia.

## Założenia i budowa systemu

System składa się z dwóch aplikacji stacjonarnych, dwóch mobilnych oraz bazy danych. Aplikacje mobilne komunikują się z usługą internetową za pośrednictwem protokołu HTTP i poprzez wymianę zapytań POST w formacie JSON. Dane zgromadzone przez usługę lokowane są w bazie danych, a aplikacja portalu administracyjnego dostarcza możliwość wglądu oraz ich edycji.

Poniżej przedstawiamy schemat systemu:

// to do obrazek

Założenia:

1. Pracownik jest związany z telefonem i kartą SIM

W bazie danych są przechowywane informacje takie jak numer IMEI oraz IMSI urządzenia którym posługuję się pracownik. Pozwala to jednoznacznie zweryfikować tożsamość osoby z której telefonu przyszło zgłoszenie.

1. Pracownik może obsługiwać tylko jedno zdarzenie w danej chwili

Powyższe założenie pozwala jednoznacznie stwierdzić ile pracownik poświęcił czasu na wykonanie zlecenia. Jednocześnie jest to zabezpieczenie przed błędami ludzkimi.

1. W przypadku zgłoszenia zawierającego niezarejestrowany znacznik, system zgłosi błąd

Z powodu jednoznacznego powiązania zdarzenia z lokalizacją niemożliwa jest obsługa zgłoszenia pochodzącego od telefonu komórkowego zawierającego identyfikator znacznika nie będący wprowadzonym do systemu.

1. Istnieją 3 grupy dostępu do systemu

System posiada rozbudowany moduł uwierzytelniający użytkowników. Stworzone zostały trzy kategorie dostępu. Każdy użytkownik posiadający login i hasło również posiada jeden z trzech poziomów uprawnień.

* ROLE\_USER – jest to dostęp pozwalający na zalogowanie się do panelu administracyjnego w trybie z ograniczonym wglądem do danych. Użytkownik ma możliwość korzystania z klienckiej aplikacji mobilnej i obsługi zdarzeń.
* ROLE\_SUPER\_USER – rola ta została stworzona z przeznaczeniem dla pracownika wprowadzającego zgłoszenie do panelu administracyjnego który jednak nie ma prawa do nadawania uprawnień innym pracownikom. Może on też korzystać z mobilnej aplikacji administracyjnej.
* ROLE\_ADMIN – jest to kategoria posiadająca wszystkie poniższe uprawnienia. Administrator ma możliwość nadawania dostępów do panelu administracyjnego.

1. Nie może być sytuacji w której na ten sam znacznik/ lokalizacje są stworzone dwa zdarzenia o statusie utworzone/rozpoczęte

Jest to spowodowane maksymalnym uproszczeniem interfejsu użytkownika aplikacji mobilnej, tak aby rejestracja rozpoczęcia/zakończenia zdarzenia odbywała się w sposób jak najbardziej intuicyjny i nie angażujący.

1. Każda lokalizacja jest jednoznacznie definiowana przez identyfikator znacznika

Przyjmujemy, że każde urządzenie kontrahenta posiada indywidualny znacznik, naklejony na obudowę.

1. Lokalizacje są trwale powiązane z kontrahentem.

Założenie to zwiększa przejrzystość prezentacji danych.

1. Wszystkie dane lokalizacji są przechowywane w bazie danych

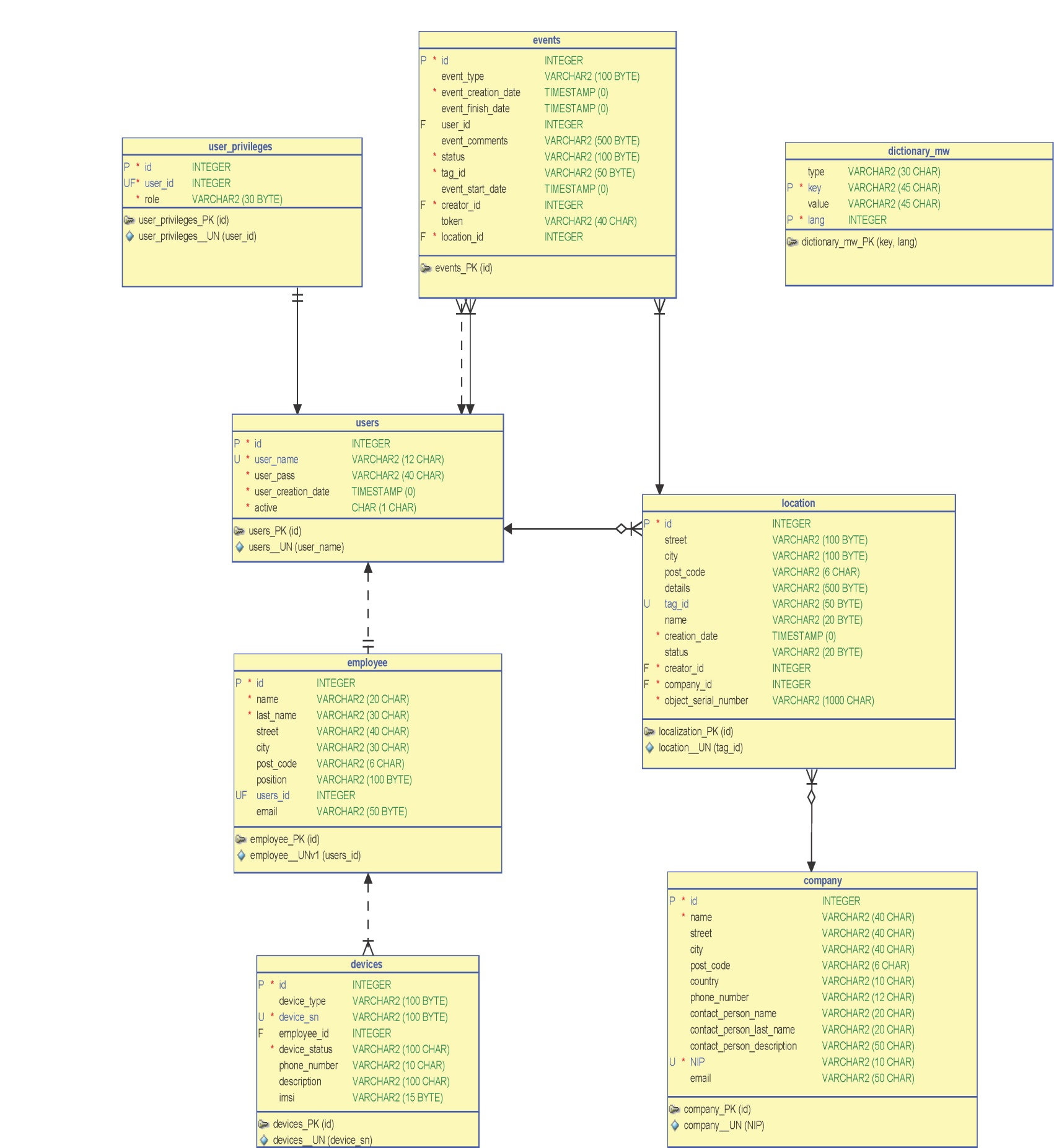
System nie dokonuje zapisu informacji charakterystycznych dla lokalizacji na znaczniku w celu uniknięcia rozproszenia i ujawnienia danych.

# Opis bazy danych

Baza danych jest podstawowym elementem opracowanego przez nas systemu. Jej zadanie jest przechowywanie danych o zdarzeniach, użytkownikach, urządzeniach, lokalizacjach oraz kontrahentach tak aby móc zapewnić poprawne działanie panelu administracyjnego oraz usługi sieciowej.

Została ona stworzona w języku SQL na serwerze Oracle 10g Express Edition. Dzięki zaawansowanym możliwościom bazy danych Oracle stworzyłem mechanizmy zabezpieczające bazę przed wprowadzeniem błędnych danych, danych niezgodnych z przyjętymi założeniami oraz powtarzających się rekordów.

Na rysunku … został przedstawiony schemat użytej w systemie bazy danych.



*Rysunek …Schemat stworzonej bazy danych*

## Opis tabel:

Poniżej zamieszczam skrótowy opis poszczególnych tabel z powyższego diagramu. Dokładny opis każdej z tabeli, ich pól został zamieszczony w dodatku A

Tabela users przechowuje dane użytkowników mających dostęp do systemu. Zawiera ona login, skrót hasła utworzony przy pomocy algorytmu sha-1, datę zarejestrowania oraz to czy jest on aktywny czy też nie.

Tabela user\_privileges przechowuje identyfikator użytkownika będący kluczem obcym z tabeli users oraz rolę przypisaną do użytkownika. Relacja pomiędzy tabelą users a user\_privileges jest relacją 1 do 1 a więc tylko jedna rola może zostać przypisana danemu użytkownikowi. W tabeli mogą być tylko przechowywane określone role. Wykaz roli wraz z ich opisem został już wcześniej przedstawiony w rozdziale o koncepcji systemu

Tabela employee jest główną tablą przechowującą informacje szczegółowe na temat pracowników zarejestrowanych w systemie. Zawiera on identyfikator użytkownika będący kluczem obcym pozwalający na określenie loginu i hasła danego pracownika. Relacja pomiędzy tabelą users a employee jest relacją 1 do 1, przez co nie możliwa jest sytuacja aby użytkownik miał przypisany więcej niż jeden login do systemu.

Tabela devices ma za zadanie przechowywanie danych urządzeń zarejestrowanych w systemie. Posiada klucz obcy do tabeli employee w relacji wiele do jednego dzięki któremu możliwy jest mechanizm przypisania urządzenia do danego pracownika. Pracownik może posiadać wiele urządzeń ale już konkretne urządzenie może zostać przypisane tylko jednemu pracownikowi. Każde urządzenie zarejestrowane w systemie musi posiadać status zgodny z przyjętą konwencją.

Tabela events zawiera dane opisujące zdarzenia rejestrowane przez usługę sieciową lub panel administracyjny. Każde zdarzenie obowiązkowo musi posiadać status, datę utworzenia, identyfikator Tagu dla lokalizacji oraz identyfikator użytkownika który zarejestrował dane zdarzenie. Tabela events posiada trzy klucze obce, dwa do tabeli users – mające za zadanie przechowywać identyfikatory użytkownika który utworzył zdarzenie oraz użytkownika który danym zdarzeniem się zajmuję. Trzecim kluczem obcym jest identyfikator lokalizacji na którą zarejestrowane jest zdarzenie.

Tabela locations jest miejscem w którym przechowywane sa dane jednoznacznie wiążące unikalny identyfikator Tagu, nazwę lokalizacji oraz numer seryjny urzadzenia. Każda lokalizacja jest powiązana w relacji jeden do wielu z tabelą company. Oznacza to, że każda lokalizacja jest jednoznacznie powiązana z firmą, jednocześnie na daną firmę może zostać zarejestrowane wiele lokalizacji.

Tabela company zawiera informacje szczegółowe opisujące kontrahentów zarejestrowanych w panelu administracyjnym. Każdy kontrahent jest jednoznacznie określona za pomocą numeru NIP.

## Integralność danych

W celu zapewnienia spójności i integralności danych przechowywanych w bazie stworzyłem szereg ograniczeń, tak aby niemożliwe było stworzenie rekordów z błędnymi wpisami.

Za pomocą klauzuli check założonej na tabelach events, devices oraz locations sprawdzam czy wpisywane statusy obiektu są zgodne z założonymi :

|  |  |
| --- | --- |
| STATUS | OPIS |
| LOCATION\_STATUS\_CREATED | lokalizacja utworzona |
| LOCATION\_STATUS\_ACTIVE | lokalizacja aktywna |
| LOCATION\_STATUS\_INACTIVE | lokalizacja nieaktywna |
| EVENT\_STATUS\_CREATED | zdarzenie utworzone |
| EVENT\_STATUS\_STARTED | zdarzenie rozpoczęte |
| EVENT\_STATUS\_SUSPENDED | zdarzenie zawieszone |
| EVENT\_STATUS\_FINISHED | zdarzenie zakończone |
| EVENT\_STATUS\_CLOSED | zdarzenie zamknięte |
| DEVICE\_STATUS\_CREATED | urządzenie utworzone |
| DEVICE\_STATUS\_ACTIVE | urządzenie aktywne |
| DEVICE\_STATUS\_INACTIVE | urządzenie nieaktywne |

*Tabela …Lista używanych statusów obiektów*

Dodatkowo stworzyłem mechanizm obejmujący tabelę events, niepozwalający na zamieszczenie w niej dwóch wpisów z tym samym TAG\_ID będących w statusie started lub created. Poniży diagram pokazuje jakie sytuacje nie mogą mieć miejsca:

|  |  |  |
| --- | --- | --- |
| TAG\_ID | STATUS |  |
| Abc | EVENT\_STATUS\_CREATED | yes.gif |
| Abc | EVENT\_STATUS\_STARTED | yes.gif |
| Abc | EVENT\_STATUS\_FINISHED | yes.gif |
| Abc | EVENT\_STATUS\_FINISHED | yes.gif |
| Abc | EVENT\_STATUS\_CLOSED | yes.gif |
| Abc | EVENT\_STATUS\_CLOSED | yes.gif |
| Abc | EVENT\_STATUS\_CREATED | no.gif |
| Abc | EVENT\_STATUS\_STARTED | no.gif |

*Tabela … Przedstawienie mechanizmu integralności danych*

## Wyzwalacze i sekwencje

W celu zapewnienia unikalności kluczy głównych stworzyłem dla każdej tabeli wyzwalacz (ang. Trigger ) oraz sekwencję które to mają na celu zadbanie o generowanie pól ID.

Poniżej zamieszczam przykładowy kod sekwencji oraz wyzwalacza :

create sequence events\_id\_seq start with 1 increment by 1;

create or replace trigger events\_insert

before insert on events

for each row

begin

if :new.id is null then

select events\_id\_seq.nextval into :new.id from dual;

end if;

end;

*Kod źródłowy … Przykład wyzwalacza oraz sekwencji tabeli events*

# Opis usługi sieciowej – Web Service

Stworzona przeze mnie usługa sieciowa jest samodzielną aplikacją zainstalowaną na serwerze której zadaniem jest nasłuchiwanie i przetwarzanie zgłoszeń napływających od aplikacji klienckich zainstalowanych na telefonach pracowników mobilnych. Można wyróżnić jej cztery główne funkcjonalności pozwalające kolejno na :

* uwierzytelnienie pracownika i umożliwienie mu dalszej pracy z aplikacją mobilną
* sprawdzenie przez aplikacje kliencką czy w bazie danych istnieją zdarzenia będące w statusie „STARTED”
* rejestrację nowego zdarzenia, lub też zamknięcie już istniejącego
* wprowadzenie do systemu nowej lokalizacji wraz z nowym identyfikatorem Tagu

Zaimplementowane rozwiązanie składa się z szeregu klas zgromadzonych w 5 modułach zapewniających poprawne działanie powyższych funkcjonalności.

* Moduł DAO
* Moduł Entity
* Moduł WebService
* Moduł WebService.Request
* Moduł WebService.Response

Obiekty DAO (ang. Data Access Object ) odpowiadają za wykorzystanie szkieletu aplikacji Hibernate 3 którego zadaniem jest poprawne mapowanie obiektów biznesowych zawierających się z module Entity na encje bazy danych.

Wraz z obiektami biznesowymi w module Entity przechowywane są pliki odwzorowań hibernate pozwalające na odpowiednie odtworzenie tabel i ich kolumn na obiekty klas entity.

Moduł web service zawiera klasy będące implementacją usługi sieciowej w rozumieniu architektury REST. Każda z nich świadczy określone funkcje dla aplikacji mobilnej.

Dwa ostatnie moduły zawierające klasy Request oraz Response przechowują obiekty będące odwzorowaniami zapytań i odpowiedzi przesyłanymi w formacie JSON pomiędzy komunikującymi się stronami.

Dodatkowo aplikacja zawiera plik konfigurujący hibernate.cfg.xml który określa informacje potrzebne do połączenia się z bazą danych a także pule połączeń czy też dodatkowe funkcję hibernate.

Poniżej opiszę każdą z wyżej wymienionych funkcjonalności a także zaprezentuje przypadki wykorzystania ich w aplikacji.

// nie wiem czy wklejać kod przykładowego odwzorowania hbm.xml i hbm.cfg.xml

## Mechanizm Uwierzytelnienia.

Mechanizm uwierzytelnienia opiera się na wysłaniu zapytania przez aplikację mobilną na odpowiedni adres usługi sieciowej. Zapytanie ma na celu ustalenie czy dany użytkownik jest zarejestrowany w systemie, oraz czy urządzenie z którego wysyła zapytanie jest przypisane właśnie do niego.

Aplikacja mobilna przesyła zapytanie na adres /nfcTimeControll/service/auth które zawiera login, hasło, numer IMEI telefonu komórkowego oraz numer IMSI będący identyfikatorem karty SIM włożonej do aparatu.

Przesłane dane pozwalają na jednoznaczne określić uprawnienia użytkownika, jego dane, to czy nie jest zablokowany przez administratora systemu oraz to czy urządzenie którym się posługuje jest zarejestrowane na jego nazwisko.

Implementując ten mechanizm wykraczający ponad sprawdzenie jedynie poprawności nazwy użytkownika oraz hasła ze względu na chęć zabezpieczenia się przed możliwością zamiany aparatów, czy też karty SIM pomiędzy pracownikami mobilnymi. Chcemy mieć pewność, że dane zdarzenie jest obsługiwane konkretnie przez pracownika będącego właścicielem telefonu.

// we wstępie teoretycznym napisać co to imei, imsi,

// prosty diagram

## Mechanizm sprawdzania statusu zdarzenia

Został zaimplementowany mechanizm pozwalający na sprawdzenie czy w bazie danych istnieją zdarzenia otwarte i przypisane już dla zgłaszającego się użytkownika mobilnego. Pozwala to na zabezpieczenie aplikacji przed błędnym działaniem i potraktowaniu zgłoszenia będącego de faco zamknięciem już istniejącego jako nowe.

Powodem takiego działania są względy technologiczne implementacji aplikacji na telefon komórkowy. Niestety poprzez wyczyszczenie podręcznej pamięci użytkownika aparatu telefonicznego możliwe jest usunięcie zgromadzonych tam danych odnoszących się do już zarejestrowanego i rozpoczętego zdarzenia.

Dzięki wyżej wymienionemu mechanizmowi telefon automatycznie uzupełnia potrzebne mu do dalszego funkcjonowania informacje.

Każde zapytanie o status będące reprezentacją klasy CRequestStatus zawiera dane potrzebne do ponownego uwierzytelnienia użytkownika i jednocześnie pozwalające na odpytanie bazy danych o zdarzenia w statusie STARTED przypisane go danego pracownika.

// jakiś Prosty diagram.

## Rejestracja nowego zdarzenia

Mechanizm rejestracji nowego zdarzenia oraz odnotowania zakończenia już istniejącego jest podstawową funkcjonalnością systemu. Usługa sieciowa nasłuchuje na adresie /nfcTimeControl/service/reg na zgłoszenia przychodzące z aplikacji mobilnych.

Każde zgłoszenie przychodzące od telefonu komórkowego zawiera wymienione już wcześniej dane pozwalające na uwierzytelnienie pracownika oraz dodatkowe informacje na temat rozpoczętego czy też zakończonego zdarzenia. Dzięki unikalnemu identyfikatorowi Tagu jednoznacznie określającego zdarzenie oraz dacie system jest w stanie wprowadzić do bazy danych nowe zdarzenie oznaczając je jednocześnie statusem „rozpoczęte”. W przypadku gdy zgłoszenie ma na celu zakończenie wcześniej już zarejestrowanego zdarzenia przesyłany jest specjalnie tworzony token jednoznacznie identyfikujący zdarzenie. Na jego podstawie zdarzenie uzyskuje status „zakończone”. Token jest tworzony w oparciu o nazwę użytkownia zajmującego się zdarzeniem, datę utworzenia oraz identyfikator Tagu.

Uproszczony schemat działania rejestracji nowego zdarzenia został przedstawiony na rysunku 1.1

/// tu stworzyć diagram w Visio na całą stronę.

Na podstawie powyższego schematu możemy zobaczyć krok po kroku jakie czynności wykonuje system przez odnotowaniem zdarzenie. Zaraz po uwierzytelnieniu użytkownika aplikacja sprawdza czy w przychodzącym zgłoszeniu przesłany jest token. Jeżeli jest, wiemy, że zdarzenie jest już w statusie „rozpoczęte” w bazie danych i należy je zakończyć poprzez ustawienie odpowiedniego statusu a także daty zakończenia.

W przypadku braku tokenu system przystępuje do kolejnego kroku jakim jest sprawdzenie czy dany identyfikator Tagu jest zarejestrowany w systemie. Jeżeli weryfikacja jest negatywna aplikacja mobilna otrzymuje odpowiedni status błędu mówiący o braku Tagu w bazie. W przeciwnym razie aplikacja sprawdzi najpierw czy istnieją zdarzenia o statusie „utworzone” dla danego Tagu, lub też utworzy nowe zdarzenie.

Każdy krok kończy się przesłaniem odpowiednio sformułowanej odpowiedzi będącej klasą CResponse która to zawiera pole informujące o statusie przetwarzanego zgłoszenia oraz dwa pola wypełniane w przypadku rejestracji zgłoszenia – komentarz do zgłoszenia utworzonego wcześniej przez panel administracyjny oraz token.

// opisać statusy – diagram zmian statusu - w rozdziale podstawy teoretyczne / założenia

// opisać w jaki sposób lokalizacja jest powiązana z Tag\_id i evntami – w rozdziale założenia teoretyczne

## WPROWADZANIE NOWEJ LOKALIZACJI I TAGU

Funkcjonalność rejestracji nowej lokalizacji jest przeznaczone dla pracowników posiadających odpowiednio wysokie uprawnienia, takie jak ROLE\_SUPER\_USER czy też ROLE\_ADMIN.

Mechanizm ten został stworzony z myślą o potrzebie łatwej rejestracji tagów bezpośrednio na miejscu w zdalnej lokalizacji na przykład podczas podpisywania nowej umowy z kontrahentem. Umożliwia ona osobie wprowadzającej nową lokalizacje wprowadzenie podstawowych danych do zapisania jej w systemie.

Tak jak w poprzednich przypadkach usługa sieciowa obsługuje zgłoszenia przychodzące na adres /nfcTimeContorl/service/admin. Każde zgłoszenie zawiera informacje na temat użytkownika służące do jego uwierzytelnienia, oraz identyfikator Tagu, datę rejestracji, nazwę lokalizacji, numer seryjny urządzenia z którym będzie powiązany Tag a także nip i nazwę firmy do której należy dana lokalizacja.

Na podstawie przesłanych danych system decyduje o swoich następnych krokach. Jeżeli identyfikator tagu:

* istnieje już w bazie – wtedy aplikacja odmawia jej rejestracji
* nie istnieje ale firma będąca jej właścicielem istnieje wtedy rejestruje nową lokalizacje
* nie istnieje, oraz firma do której należeć będzie lokalizacje nie istnieje wtedy rejestruje najpierw nową firmę a następnie lokalizację.

// w wstępie teoretycznym opisać powiązanie lokalizacji z Tagiem i company

Uproszczony mechanizm działania tej funkcjonalności przedstawia rysunek 1.2

// narysować w Visio schemat.

Ze schematu możemy odczytać jak wygląda algorytm tworzenia nowej lokalizacji. Na początku zostaje potwierdzona tożsamość użytkownika. Następnie system sprawdzi czy dany Tag istnieje już w bazie danych. Jeżeli istnieje użytkownikowi zostanie zwrócony odpowiedni status. W przeciwnym wypadku baza danych zostanie odpytana czy dla przychodzącego w zgłoszeniu numeru NIP istnieje firma zarejestrowana w systemie. Jeżeli tak, nowa lokalizacja zostanie stworzona i podpięta pod istniejącą już firmę, w przeciwnym wypadku system zarejestruje nową firmę, lokalizację a następnie zwróci użytkownikowi odpowiedni status opisujący pomyślne zakończenie procesu.

## Opis aplikacji klienckiej na telefon komórkowy

Przeznaczeniem aplikacji klienckiej dla pracownika mobilnego jest pełnić funkcję intuicyjnego i prostego w obsłudze interfejsu do systemu monitorowania stanu realizacji zleceń.

Założenia

Główne założenia które stanowiły podstawę przy projektowaniu tej części systemu to możliwie jak największa szybkość wywołania i działania aplikacji, oszczędność baterii, łatwość w obsłudze i przejrzysta dla użytkownika prezentacja danych.

Uproszczony opis zadań powierzonych aplikacji na telefon zawiera w sobie:

1. automatyczny odczyt zbliżonego znacznika w sytuacji w której aplikacja jest włączona
2. zgłoszenie się do odczytu znacznika zgodnego z aplikacją z poziomu menu systemu operacyjnego
3. uwierzytelnienie użytkownika i automatyzacja logowania z zachowaniem oczekiwanej kontroli tożsamości
4. Komunikacja z użytkownikiem w oparciu o graficzny interfejs
5. Duża szybkość reakcji na działania użytkownika
6. Komunikacja z serwerem w celu obsługi odczytanego znacznika

W celu maksymalnego, możliwego odciążenia aplikacji na telefon, zgodnie z ideologią pisania aplikacji na powyższy system (DOWÓD), cała logika systemu zrealizowana jest w aplikacji serwerowej, telefon z kolei formułuje zapytania i przetwarza odpowiedzi. Komunikacja w architekturze REST (Representational State Transfer – literatura) opiera się o zapytania w formie JSON (JavaScript Object Notation). Bezpośrednio oznacza to, że aplikacja zgłasza do serwera zapytanie w momencie gotowości do przejścia w inny stan działania (np. przedstawienie obecnego stanu zgłoszenia). Forma tego zapytania jest ściśle znana i wysyłana w sposób niejawny w formacie JSON. Zaletą tego formatu jest ogólnodostępność i niezależność od języka programowania, idące jednak w parze z kompletnym wsparciem.

Komunikacja pomiędzy usługą internetową a użytkownikiem aplikacji

Na potrzeby komunikacji między aplikacją mobilną i usługą internetową stworzony został zestaw statusów przenoszonych w obiekcie CResponse odpowiedzi. Do zadań aplikacji należy możliwie jak najbardziej nieingerencyjne obsłużenie danego statusu, lub poinformowanie użytkownika o zaistniałej sytuacji.

Do sytuacji takich należą:

* komunikaty o braku dostępu:

ACCESS\_DENIED,

WRONG\_PASSWORD\_ERROR,

USER\_NOT\_FOUND\_ERROR

* komunikat o nie odnalezieniu danego znacznika w bazie danych:

TAG\_NOT\_FOUND\_ERROR

* komunikat informujący o fakcie uprzedniego rozpoczęcia realizacji innego zgłoszenia (przy próbie rozpoczęcia innego):

EVENT\_ALREADY\_STARTED\_ERROR

* \*\*\*\*\*\*\*\*\*\*\*\*\*\*

W każdej z tych sytuacji aplikacja informuje użytkownika i dokonuje przekierowania interfejsu w celu ułatwienia rozwiązania sytuacji (przejście do ekranu logowania, wyświetlenie stanu obecnie wykonywanego zlecenia itp.) GODZINA ROZPOCZĘCIA

Tryby uruchomienia

Z przyczyn praktycznych aplikacja może być wywołana dwojako: przez użytkownika z poziomu menu systemu operacyjnego Android, lub automatycznie w wyniku zbliżenia telefonu do znacznika. Metody te różnią się cyklem wywołań intencji i działaniami wykonywanymi w tle.

W pierwszym wypadku do obsługi znacznika stosowany jest tzw. foreground dispatching, **opisany dokładniej w podrozdziale Dobór aplikacji do obsługi znacznika**. Niestety metoda ta nie działa zawsze idealnie i czasem system operacyjny pomimo tego faktu pyta się użytkownika o to której aplikacji użyć do obsłużenia danego znacznika, jednak w obecnej wersji aplikacji są to przypadki sporadyczne.

Dodatkowo aplikacja w pierwszej fazie uruchomienia sprawdza czy posiada zapisane w banku UserPreferences dane logowania i jeśli nie może ich znaleźć, oferuje użytkownikowi wprowadzenie ich bądź opuszczenie aplikacji. Ponieważ przy każdym odczytaniu znacznika dane zdarzenia transmitowane do serwera opatrzone są kompletem danych identyfikacyjnych (z racji konstrukcji aplikacji typu RESTful nie posługujemy się sesjami dla użytkowników tylko wymianą zapytań i odpowiedzi), w sytuacji zgłoszenia jednego ze statusów braku dostępu zachodzi przekierowanie do aktywności wprowadzania danych logowania.

Zmiana danych logowania jest też dostępna z poziomu menu aplikacji w każdej chwili. Hasło do logowania przechowywane jest w aplikacji w formie skrótu SHA-1, ponieważ…

W tym wariancie aplikacja dokonuje również aktualizacji stanu obecnie obsługiwanego przez danego użytkownika zdarzenia, jeśli takie istnieje. Dzięki temu nawet w sytuacji odinstalowania i ponownego zainstalowania, zmiany danych logowania lub usunięcia danych użytkownika, ten ostatni jest zawsze poinformowany o swoim stanie zarejestrowanym w systemie.

Sytuacją odmienną jest zbliżenie telefonu do znacznika w stanie spoczynku (przy wyłączonej aplikacji). System operacyjny wyświetla wtedy monit z zapytaniem o aplikację która powinna zostać oddelegowana do obsłużenia znacznika, jest to tak zwany system Entent dispatching, **który również zostanie przybliżony w podrozdziale Dobór aplikacji do obsługi znacznika**.

W sytuacji takiej aplikacja uruchamia się tylko w celu odnotowania zdarzenia i po zakończeniu tego procesu i poinformowaniu użytkownika automatycznie się zamyka (nie wymaga to interakcji ze strony użytkownika i nie zajmuje pamięci systemu). Oczywiście w sytuacji podania niepoprawnych danych logowania, interfejs wyświetla stosowny komunikat i prowadzi do aktywności wprowadzenia danych.

Odczyt znacznika

Co prawda standard NFC dla znaczników NFC Forum Type 2 zapewnia prędkość transmisji 106 kbit/s, a sam znacznik posiada zaledwie 384 bitów pamięci na dane użytkownika, na drodze testów okazało się, że zapis i odczyt jakichkolwiek danych na samym znaczniku mija się z celem. Przyczyna leży w niewielkiej pojemności pamięci wybranego przez nas znacznika i unikaniu rozpraszania danych. Przechowywanie kompletu informacji w bazie danych zapewnia ich integralność. **(NFC Forum Tag Type 2)**

System operacyjny w momencie wykrycia znacznika w obrębie pola elektromagnetycznego wytwarzanego przez telefon pobiera z niego unikatowy identyfikator i wiadomości NDEF i umieszcza w intencji, która w dalszej kolejności przekazywana jest do odpowiedniej aplikacji w celu ich przetworzenia. W przypadku naszej aplikacji przetwarzamy jedynie unikatowy identyfikator znacznika, który przesyłany jest do bazy danych. Odczyt wiadomości NDEF ma miejsce jedynie do konsoli w celach testowych. **(ANDROID API)**

Dobór aplikacji do obsługi znacznika

W wersji Android 2.3.4 komunikacja NFC doczekała się serii modyfikacji i rozszerzeń które pozwoliły między innymi w bardziej precyzyjny sposób delegować odpowiednią aplikację do obsługi danych zawartych na znaczniku. **(KSIĄŻKA INTERNETOWA ANDROID)**

W tym celu stworzono dwa systemy wywoływania aplikacji.

Pierwszym z nich jest intent dipatch składający się z trzech następujących po sobie filtrów *akcji*, które mają na celu możliwie jak najbardziej zawężać pulę zainstalowanych aplikacji tak, aby użytkownik telefonu nie musiał podejmować żadnych decyzji tylko był obsługiwany automatycznie. Jeśli dane kryterium spełnia więcej niż jedna aplikacja, wyświetlany jest monit z prośbą o wybór odpowiedniej. Poniżej przedstawię pokrótce wspomniane typy akcji:

android.nfc.action.NDEF\_DISCOVERED

Powyższa akcja opiera się na analizie typu wiadomości zamieszczonych na znaczniku. Jeśli dany typ wiadomości widnieje w pliku AndroidManifest.xml dowolnej aplikacji, na tym kroku przerywana jest analiza i podejmowany wybór

android.nfc.action.TECH\_DISCOVERED

W sytuacji gdy poprzedni filtr zawiedzie, sprawdzany jest technologia w jakiej wykonany został znacznik.

android.nfc.action.TAG\_DISCOVERED

Akcja ta wywoływana jest w sytuacji gdy powyższe filtry zawiodły, lub znacznik jest nieznanego typu.

Druga z metod obsługi odczytanego znacznika, tzw. foreground dispatching, wymaga od programisty samodzielnego wywołania i kontroli w cyklu życia aplikacji. Opiera się ona na narzuceniu najwyższego priorytetu obsługi znacznika aplikacji uruchomionej na pierwszym planie. Oczywiście znacznik spełniać musi wymogi określone w filtrach typu danych i technologii jego wykonania.

Jak było wspomniane wcześniej, aplikacja kliencka wykorzystuje obie te metody.

Wprowadzenie danych i komunikacja z usługą internetową

Weryfikacja autentyczności użytkownika, z powodu upatrywanych zagrożeń w postaci ludzkiej pomysłowości i złej woli, nie może przebiegać z użyciem samej nazwy użytkownika i hasła. Z tego powodu do grupy danych którymi musi się uwierzytelnić aplikacja kliencka, doszły numer IMEI oraz IMSI (rozwinięcie skrótów), dzięki czemu upewniamy się, że użytkownik o danym identyfikatorze zgłasza swoją aktywność przy pomocy swojego telefonu (numer IMEI) oraz swojej karty SIM (numer IMSI), co eliminuje zagrożenia płynące z wymiany telefonami i danymi autoryzacyjnymi pomiędzy użytkownikami systemu.

|  |  |
| --- | --- |
| Parametr | Opis |
| Identyfikator użytkownika | Przydzielony przez administratora systemu, unikatowy w skali systemu |
| Hasło | Spełniające wymogi bezpieczeństwa **WYMOGI** |
| Numer IMEI | Unikatowy, piętnasto lub siedemnastocyfrowy identyfikator urządzenia GSM lub UMTS, nie jest związany z konkretnym użytkownikiem. |
| Numer IMSI | (IMSI Guidelines v12!!) Unikatowy numer identyfikujący kartę SIM. Stworzony na potrzeby identyfikacji abonenta na drodze radiowej w systemach GSM i UMTS. |

Powyższy komplet danych identyfikuje użytkownika i telefon w sposób jednoznaczny.

Numery IMSI i IMEI nie są zapisywane nigdzie w pamięci ani ustawieniach aplikacji, tylko pobierane poprzez API systemu operacyjnego każdorazowo przy uruchomieniu aplikacji.

Dodatkowo dane przesyłane do serwera są poddane działaniom kryptograficznym w postaci szyfrowania symetrycznego AES 256bit (hmmm???), dzięki czemu użytkownik nie może z poziomu innego terminala *udawać*, że jest użytkownikiem systemu.

Działanie aplikacji

Po pierwszym uruchomieniu aplikacji użytkownik musi wprowadzić nazwę użytkownika i hasło w celu zalogowania w systemie. Jeśli proces uwierzytelnienia przebiegnie pomyślnie, dane zostaną zapisane w banku danych aplikacji systemu Android i będą od teraz używane do rejestracji zdarzeń w systemie aż do ich zmiany lub usunięcia. Wspomniane opcje dostępne są zawsze z poziomu menu podręcznego.

Główny ekran aplikacji wyświetla tekstowo i graficznie obecny stan aktywności danego użytkownika. Jest to „W trakcie zlecenia…” lub „Oczekiwanie” w sytuacji w której żadne zlecenie nie jest obecnie obsługiwane. Jeśli zlecenie zostało rozpoczęte, dodatkowo wyświetlany jest krótki komentarz wprowadzony przez operatora który wprowadził dane zlecenie do systemu **oraz godzina rozpoczęcia**. Informację o stanie użytkownika aplikacja pozyskuje w procesie synchronizacji z usługą internetową tuż po uruchomieniu aplikacji.

W sytuacji chęci rozpoczęcia zlecenia, użytkownik musi jedynie zbliżyć telefon do znacznika przyklejonego na obudowie urządzenia. Na podstawie kompletu danych użytkownika, identyfikatora i daty systemowej formułowane jest zapytanie wysyłane do usługi internetowej. Data systemowa wysyłana jest w celu porównania z datą aplikacji serwerowej w sytuacji w której pracownik mógłby mieć zastrzeżenia co do stanu rozliczenia z pracodawcą. W wiadomości zwrotnej przekazywany jest unikatowy token danego zdarzenia, niezbędny przy zgłaszaniu jego zakończenia. Rozpoczęcie zdarzenia sygnalizowane jest wibracją, dźwiękiem i odpowiednim oknem dialogowym, zawierającym komentarz wprowadzony przez operatora.

Z poziomu ekranu opcji istnieje możliwość wyłączenia wszystkich metod sygnalizacji prócz okna dialogowego.

Zakończenie realizacji zlecenia wykonuje się analogicznie, z tym, że wymaga to również potwierdzenia poprzez stosowne okno dialogowe.

Usiłowanie rozpoczęcia realizacji drugiego zlecenia przed zamknięciem poprzedniego zakończy się fiaskiem, gdyż logika aplikacji serwerowej na to nie dopuści. Użytkownik zostanie poinformowany o tym fakcie poprzez stosowne okno dialogowe.

Również w sytuacji wystąpienia błędu komunikacji lub wyjątku, użytkownik zostanie o tym poinformowany, a aplikacja powróci do stanu oczekiwania.

Wszystkie procesy komunikacji z usługą internetową przeniesione są do wątku roboczego, toteż problemy z połączeniem lub wydłużony czas realizacji zapytania nie wpływa na interfejs graficzny, dzięki czemu nawet przy wystąpieniu problemów z zasięgiem nie dochodzi do zjawiska tzw. braku odpowiedzi ze strony aplikacji.

Wpływ na zużycie baterii

Podczas testów wydajnościowych stwierdzono, ze ciągłe działanie aplikacji nie ma praktycznie żadnego wpływu na zużycie baterii. To samo tyczy się stale włączonego modułu NFC (zużycie poniżej 1% baterii).

## Opis aplikacji administracyjnej na telefon komórkowy

Aplikacja administracyjna używana jest do rejestracji lokalizacji przez administratora systemu. Za jej pomocą wiąże on dany Tag z klientem i konkretnym urządzeniem.

W chwili obecnej, z przyczyn finansowych (nie mamy dostępu do nieograniczonych idlo, nie zapisujemy nic na Tag w postaci „tylko do odczytu” i opieramy działanie systemu o unikatowe identyfikatory tagów, które powiązane są w bazie z daną lokalizacją.

# Opis panelu administracyjnego

Panel administracyjny jest ostatnim a zarazem najbardziej rozbudowanym elementem systemu. Pełni kluczową rolę w zarządzaniu naszym rozwiązaniem. …

Użytkownikiem panelu administracyjnego są osoby zarządzające czasem pracy pracowników mobilnych. Dzięki dostarczonym funkcjonalnością mogą bez trudu kontrolować czas oraz stopień zaawansowania pracy. Aplikacja udostępnia również wgląd do informacji zgromadzonych w bazie danych. Dzięki niej możliwe jest edytowanie wpisów, oraz dodawanie nowych do już istniejących.

## Szczegóły implementacyjne :

Aplikacja webowa do poprawnego działania wymaga obecności dowolnego serwera aplikacyjnego zgodnego ze standardem Java Enetrpriese Edition. Musi być on kompatybilny z technologią Servletów, oraz Java Server Pages. W naszym przypadku zdecydowałem się na użycie serwera aplikacyjnego Apache Tomcat w wersji 6.0.32. Dzięki niemu mogłem stworzyć warstwę prezentacji opartą na stronach JSP współpracujących z bibliotekami JSTL (ang. Java Standard Tag Library ). Dzięki nim możliwe jest oddzielenie kodu źródłowego od warstwy prezentowanych danych.

Środowiskiem programistycznym w którym został wykonany panel jest :

Java SDK 1.6.0\_26

Tomcat 6.0.32

Oracle 10g XE wraz z sterownikiem jdbc w wersji 6

Eclipse Helios wraz z zainstalowanym pakietem Web Tools Platform

Dodatkowo, panel administracyjny został wykonany w oparciu o szkielet aplikacji Spring w wersji 3.0.1, pakiet Spring Security w wersji 3.0.5 oraz technologię ORM (ang. Object Relational Mapping ) którą dostarcza Hibernate 3. W celu zapewnienia poprawności logowania zdarzeń aplikacji oraz ewentualnych wyjątków zawarty został również Framework Log4j.

Aplikacja webowa implementuje wzorzec MVC – Model Widok Kontroler który to został już opisany przeze mnie we wcześniejszej części pracy. Poniżej prozentuje opis struktury stworzonego przeze mnie panelu.

### Warstwa biznesowa:

Warstwę biznesową tworzą moduły zawierające klasy biorące udział w tworzeniu logiki aplikacji. Ich lista została przedstawiona poniżej.

* net.nfc.web.controller
* net.nfc.web.forms
* net.nfc.web.mailservice
* net.nfc.web.service
* net.nfc.web.validator

Moduł net.nfc.web.controller zawiera klasy odpowiadające za odpowiednie mapowanie zapytań przychodzących z warstwy prezentacji na odpowiednie zasoby. Zawarta jest w nich logika pozwalająca na przetwarzanie danych z formularzy oraz przesyłanie ich dalej do odpowienich obiektów zajmujących się ich walidacją lub zapisem do bazy danych.

net.nfc.web.form grupuje klasy potrzebne do obsługi formularzy wyświetlanych na stronach JSP.

net.nfc.web.mailservice odpowiada za kasy potrzebne do obsługi wysyłki wiadomości elektronicznych mających na celu informowanie pracowników o nowo utworzonych zdarzeniach.

net.nfc.web.service jest modułem który znajduję się na granicy pomiedzy warstwą modelu – czyli utrwalania danych a warstwą biznesową. Jego zadaniem jest przesyłanie obiektów do odpowiednich klas DAO (ang. Data Access Object ) tak aby te zapisały informacje w bazie danych.

net.nfc.web.validator zawiera klasy odpowiadające za poprawność danych przesyłanych w formularza przez użytkownika końcowego. To jego zadaniem jest powstrzymanie przed wpisaniem do bazy niepoprawnych wartości czy też zadbanie o wypełnienie wszystkich obowiązkowych pól.

### Warstwa utrwalania danych

Warstwa ta odpowiada za poprawne pobranie, uaktualnienie lub zapis obiektów do encji bazy danych. Za połączenie z bazą danych oraz zarządzenie transakcjami i sesjami odpowiada pakiet Hibernate. W skład tej klasy wchodzą dwa moduły :

* net.nfc.db.dao
* net.nfc.db.entity

Paczka net.nfc.db.dao zawiera klasy bezpośrednio kontaktujące się z bazą danych. Są w nich zawarte odpowiednie metody odpowiedzialne za pobieranie rekordów, listowanie danych, uaktualnianie bądź też wprowadzanie nowych informacji. Obiekty DAO również odpowiadają za filtrowanie wyników. ,,,

Moduł net.nfc.db.entity zawiera klasy mapowane

Zakładki :

Możliwe jest wyświetlenie listy wszystkich zdarzeń zarejestrowanych przez system, oraz podejrzenie szczegółowych danych odnośnie każdego z nich. Każde zdarzenia może zostać