POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

Katedra
Informatyki i Automatyki
Politechnika Rzeszowska

Bazy danych

Projekt

pt.: „Forum internetowe dla studentów Politechniki Rzeszowskiej”

Data wykonania: 05.12.2020

Grupa: L2
Rafał Magryś

POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

Katedra
Informatyki i Automatyki
Politechnika Rzeszowska

Spis treści

1. Cel pracy

Celem pracy jest stworzenie bazy danych forum internetowego dla studentów. Forum to będzie przypominało forum twitterowe, na których każdy zalogowany użytkownik będzie mógł dodawać posty w każdej uczelnianej sprawie. Baza danych będzie realizowała podstawowe funkcje jak zwracanie użytkowników, kategorii, postów, komentarzy. Dodatkowo użytkownicy mogą nawzajem oceniać swoje wypowiedzi, a te najbardziej trafne mogą być nagrodzone przez moderatora medalem.

Ten problem najłatwiej rozwiązać w relacyjnej bazie danych, gdyż przechowuje ona powiązania encji, co ułatwia pracę w taki sposób, że w prosty sposób można sprawdzić jaki użytkownik dodał jaki komentarz itp.

2. Przebieg Pracy

1.1. Określenie projektu

Określenie tematyki i zakresu projektu,przedstawienie, zagadnień związanych z tematem

Planuję wykonać bazę danych forum internetowego dla studentów Politechniki Rzeszowskiej. Będzie to forum typowo do uzyskania pomocy w różnych kwestiach, przykładowo mogą to być korepetycje, pomoc w projektach, odnajdywanie się na uczelni itp. Umożliwia on użytkownikom dodawać posty które będzie można oznaczyć tagami, komentować je i oceniać innych użytkowników i ich komentarze.
Na czele forum będą stali admini. Specjalni użytkownicy o największych prawach do edycji, usuwania i edytowania.
Porządku będą pilnować moderatorzy czyli specjalnie powołani użytkownicy, którzy mają uprawnienia do blokowania użytkowników postów i komentarzy oraz dawania nagród za liczne trafne odpowiedzi lub aktywność.

Określenie funkcji bazy danych i ich priorytetu

Bazy danych są w dzisiejszym czasie jednym z głównych miejsc przechowywania informacji. W tym projekcie baza danych będzie nie tylko przechowywać kategorie, tematy posty i komentarze ale również oceny użytkowników i medale przyznane przez moderatorów za celne odpowiedzi i niesienie wsparcia. Każdy obiekt będzie miał daty oraz inne elementy pozwalające łatwiejszą obsługę danych.

Wybór technologii i typu bazy danych do zrealizowania projektu

Projekt planuję wykonać przy użyciu relacyjnej bazy danych MySql.
Jest to darmowa baza danych która jest dosyć popularna. Jest to baza danych która cieszy się dużym wsparciem i jest wydajna.

Wybór narzędzi do zrealizowania projektu

Do zarządzania bazą danych wykorzystam MySql Workbench któro jest dosyć popularnych narzędziem. Do graficznego zaprezentowania bazy danych planuję użyć MySQL Workbench.

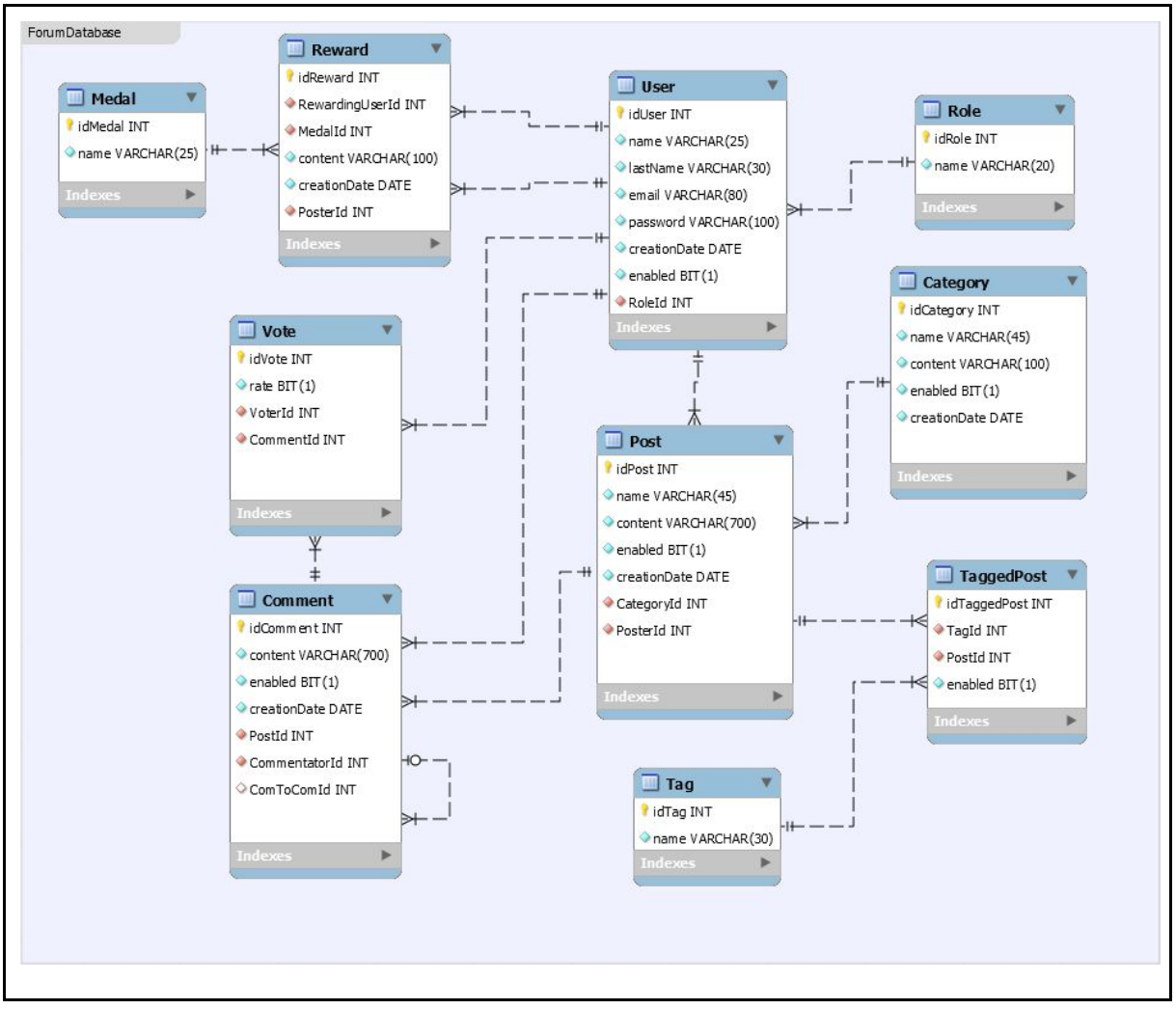
Prezentacja przygotowanego repozytorium

Repozytorium na githubie w którym będzie przechowywany projekt bazy
https://github.com/rmagrys/StudentForumDB

1.2. Prezentacja diagramu DB

Prezentacja diagramu bazy danych

Poniższy diagram związków encji prezentuje w jaki sposób moja baza danych będzie przechowywać informacje



Opis tabel bazy danych i ich funkcji

Tabela User

Jest to podstawowa tabela przechowująca podstawowe informacje o użytkowniku forum, który zależnie od uprawnień będzie mógł operować różnymi funkcjami na forum. Jest w relacji wiele do jednego z tabelą Role gdyż wielu użytkowników może posiadać tą samą rolę

Tabela Role

Przechowuje informacje o dostępnych rolach użytkowników. Docelowo będą to : administrator, moderator i użytkownik.

Tabela Reward

Przechowuje informacje o medalach jakie dostali użytkownicy za zasługi na forum. Jest dwukrotnie w relacji wiele do jednego z tabelą User gdyż użytkownik może dać wiele nagród, oraz inny użytkownik może otrzymać wiele nagród. Występuje też relacje wiele do jednego tabeli Medal, gdyż wiele nagród może posiadać ten sam medal.

Tabela Medal

Przechowuje informacje o medalach które można obdarować danego użytkownika przez moderatora

Tabela Category

Przechowuje informacje o kategorii w jakiej będą umieszczane posty. Przykładowo będzie można stworzyć kategorię - korepetycje lub projekty itd. Daje to możliwość lepszego układu informacji na forum. Przewiduję, że kategorie będzie mógł tworzyć tylko moderator lub administrator

Tabela Post

Przechowuje informacje o poście dodanym przez użytkownika. Przykładowo student będzie potrzebował pomocy w konkretnym projekcie, np z Metod Numerycznych, to bedzię mógł taki post stworzyć. Tabela jest w relacji wiele do jednego z tabelą User, gdyż użytkownik może posiadać wiele postów, oraz relacja wiele do jednego z tabelą TaggedPost, gdyż każdy post będzie mógł mieć przypiętą dowolną ilość tagów, które mają możliwość dezaktywacji

Tabela TaggedPost

Przechowuje informacje który post posiada jaki Tag, jest to tabela pomocnicza uzyskana z rozbicia relacji wiele do wielu między tabelami Tag i Post, gdyż wiele postów może posiadać ten sam tag, a ten sam tag może być oznaczony w wielu postach. Dodana jest możliwość dezaktywacji tagu w razie pomyłki, lecz jest możliwość jego przywrócenia

Tabela Tag

Przechowuje informacje o tagach dodanych przez użytkowników. Tagi te mogą być wielokrotnie wykorzystywane w różnych postach, co ułatwi wyszukiwanie “po tagu”

Tabela Comment

Przechowuje informacje o komentarzach zawartych w danym poście. Jest w relacji wiele do jednego z tabelą User, gdyż użytkownik może posiadać wiele postów. Kolejną relacją wiele do jednego jest Post, gdyż post może posiadać wiele komentarzy. Ostatecznie tabela Comment jest w relacji wiele do jednego z samą sobą, gdyż komentarz też może zostać “skomentowany” i może być ich wiele.

Tabela Vote

Jest to tabela która przechowuje informacje o ocenach użytkowników w danym komentarzu, czyli w skrócie czy jest celny czy beznadziejny.
Jest w relacji wiele do jednego z tabelą User gdyż każdy użytkownik może mieć wiele ocen, oraz w relacji wiele do wielu z tabelą Comment gdyż komentarze mogą mieć wiele ocen.

Prezentacja opisów bazy danych

Baza umożliwia zwykłemu użytkownikowi:

- przeglądanie kategorii, postów i komentarzy innych użytkowników
- sprawdzania informacji o poprawności informacji poprzez indywidualne oceny użytkowników
- sprawdzanie nagród które otrzymał użytkownik za zasługi
- dodawanie otagowanych postów i komentarzy
- oceniania komentarzy użytkowników
- edycji swojego konta i usunięcia go w razie potrzeby

Dodatkowo:

1. admin
    - może powoływać nowych moderatorów i ma pełne uprawnienia do dodawania , usuwania i banowania użytkowników.

2. moderator
    - ma mniejsze uprawnienia od admina, może dodawać kategorie, banować użytkowników, posty komentarze,

Prezentacja problemów w realizacji

Problemem w realizacji tej bazy danych może być duża ilość relacji pomiędzy encjami. Trzeba będzie być bardzo ostrożnym by nie pogubić się przy dodawaniu i edycji.

1.3. Prezentacja SQL

Prezentacja wykonania bazy danych

Bazę danych wygenerowałem przy użyciu skryptu napisanego w MySQL
Generacja na podstawie diagramu zakończyła się niepowodzeniem dlatego postanowiłem wykonać to własnoręcznie.

Zaczynam od zdropowania wszystkich tabel które mogłyby mieć taką samą nazwę jak moje

```
DROP TABLE IF EXISTS `db`.`TaggedPost` ;
DROP TABLE IF EXISTS `db`.`Reward` ;
DROP TABLE IF EXISTS `db`.`Medal` ;
DROP TABLE IF EXISTS `db`.`Vote` ;
DROP TABLE IF EXISTS `db`.`Comment` ;
DROP TABLE IF EXISTS `db`.`Post` ;
DROP TABLE IF EXISTS `db`.`Category` ;
DROP TABLE IF EXISTS `db`.`User` ;
DROP TABLE IF EXISTS `db`.`Role` ;
DROP TABLE IF EXISTS `db`.`Tag` ;
```

Następnie kod tworzący tabele, oto kilka z nich. Reszta znajduje się na moim repozytorium na GitHubie

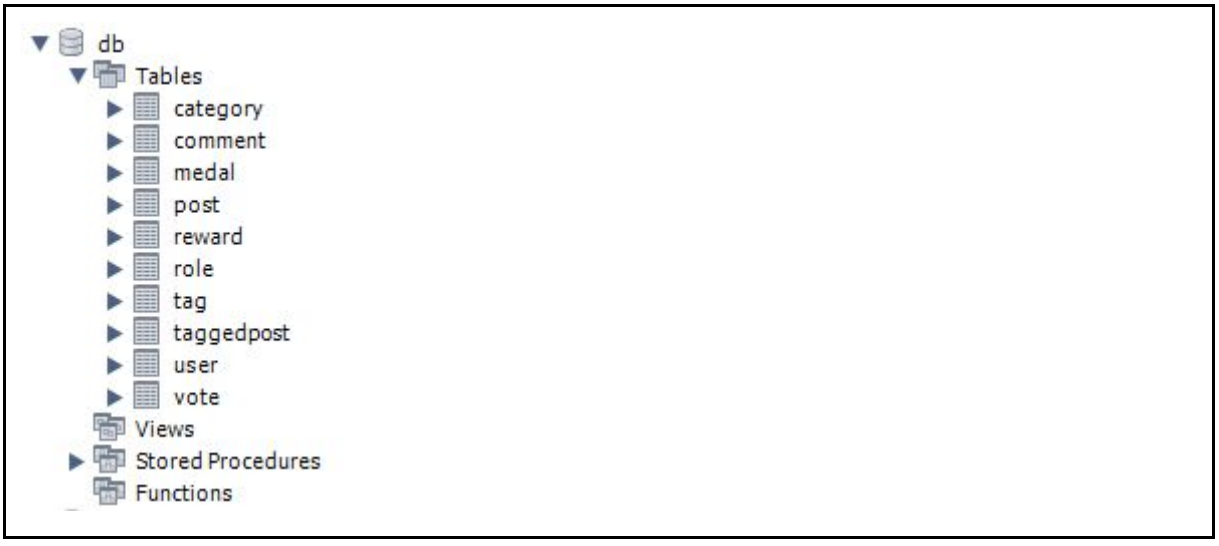
Poniżej przedstawione tworzenie dwóch tabel zaczynając od tej nadrzędnej Role, gdyż klucz obcy w tabeli User odnosi się do tabeli Role

```
CREATE TABLE IF NOT EXISTS `db`.`Role` (
  `idRole` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(20) NOT NULL,

  PRIMARY KEY (`idRole`))
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `db`.`User` (
  `idUser` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(15) NOT NULL,
  `lastName` VARCHAR(25) NOT NULL,
  `email` VARCHAR(80) NOT NULL,
  `password` VARCHAR(100) NOT NULL,
  `creationDate` DATE NOT NULL,
  `enabled` BIT(1) NOT NULL,
  `RoleId` INT NOT NULL,
  PRIMARY KEY (`idUser`),
  FOREIGN KEY (`RoleId`)
    REFERENCES `db`.`Role` (`idRole`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

Ostatecznie tabele w bazie prezentują się następująco



Prezentacja funkcji DB realizowanych przez SQL

W mojej bazie danych można dodawać użytkowników o dowolnej roli, oto procedura dodawania użytkownika

```
delimiter $$

drop procedure if exists `db`.`create user`;

CREATE PROCEDURE `db`.`create user`(
  givenName VARCHAR(15),
  givenLastName VARCHAR(25),
  givenEmail VARCHAR(80),
  givenPassword VARCHAR(100),
  givenRole VARCHAR(20)
)
BEGIN


    IF NOT EXISTS ( select email from db.user where db.user.email = givenEmail ) then
        insert into db.user(name, lastname, email, password, creationDate, enabled, roleId) value
        (givenName, givenLastName, givenEmail, MD5(givenPassword), curdate(), true,
        (SELECT idRole from db.role where name = givenRole));
    END IF;

END $$
```

Dodaje ona użytkownika sprawdzając uprzednio czy podany email już istnieje, gdyż zakładam że email będzie służył do logowania. Hasło jest hashowane

Przykładowe wywołanie procedury:

```
call db.createUser('rafal', 'magrys', 'raaff@raf.com', 'haslo', 'ADMIN');
```

A oto wynik wywołanej funkcji

idUser	name	lastName	email	password	creationDate	enabled	RoleId
49	rafal	magrys	raff@raf.com	207023ccb44feb4d7dadca005ce29a64	2020-11-25	1	1

Inna procedura służąca do wypisania wszystkich użytkowników o określonej roli

```
delimiter $$

drop procedure if exists `db`.`getAllUsersByRole`;

CREATE PROCEDURE `db`.`getAllUsersByRole`(
    givenRole varchar(20)
)
BEGIN

    select db.user.idUser, db.user.name, db.user.lastName,db.user.email,db.user.creationDate, db.role.name
    from db.user
    join db.role
    on db.user.RoleId = db.role.idRole
    where RoleId = (select idRole from db.Role where db.Role.name = givenRole) and enabled = 1
    order by db.user.creationDate desc;

END $$
```

Przykładowe wywołanie procedury

```
call db.getAllUsersByRole('USER');
```

A oto wynik wywołania (pokazana jest niewielka część)

▶	48	rafal	magrys	aff@af.com	2020-11-21	USER
	47	rafal	magrys	af@af.com	2020-11-21	USER
	36	Józef	Andrzejewski	józef.andrzejewski@158843.pl	2020-11-14	USER
	39	Oliwia	Szewczyk	oliwia.szewczyk@158843.pl	2019-06-18	USER
	17	Sylwester	Sroka	sylwester.sroka@158843.pl	2018-07-11	USER
	19	Lech	Sroka	lech.sroka@158843.pl	2018-06-14	USER

Aplikacja daje możliwość blokowania i odblokowywania zawartości, będą zajmować się tym proste procedury typu tej poniżej:

```
drop procedure if exists `db`.`changeUserState`;

delimiter $$

CREATE PROCEDURE `db`.`changeUserState`(
    dataId INT,
    enabledValue BIT(1)
    )
BEGIN

    if exists (select * from db.user where idUser = dataId) then
        UPDATE user
        SET enabled = enabledValue
        WHERE idUser = dataId;
    end if;

END $$
```

Przykład wywołania

```
call db.changeUserState(15,1);
```

Jak można zauważyć w po użyciu procedury select użytkownik został zbanowany

idUser	name	lastName	email	password	creationDate	enabled	RoleId
10	Franciszka	Michalik	franciszka.michalik@158843.pl	ff12bbd8c907af067070211d87bdf098...	2015-05-27	0	2
11	Olga	Urbaniak	olga.urbaniak@158843.pl	ff12bbd8c907af067070211d87bdf098...	2019-08-03	1	2
12	Ferdynand	Murawski	ferdynand.murawski@158843.pl	ff12bbd8c907af067070211d87bdf098...	2006-07-18	1	2

Aplikacja daje możliwość dodawania postów i komentarzy przez użytkowników, w ich realizacji pomagają proste procedury, oto jedna z nich

```
drop procedure if exists  `db`.`createPost`;
DELIMITER $$

CREATE PROCEDURE `db`.`createPost` (
  givenName VARCHAR(45) ,
  givenContent VARCHAR(500) ,
  givenCategoryName VARCHAR(45),
  givenPosterMail VARCHAR(45)
)
BEGIN

    declare userId int;
    declare categoryId int;

    select idCategory INTO categoryId from category where category.name = givenCategoryName;
    select idUser into userId from user where user.email = givenPosterMail;

    if (userId and categoryId) is not null  then
        insert into post(name, content, enabled, creationDate, CategoryId, PosterId) value
        (givenName, givenContent, true, curdate(), categoryId, userId);
    end if;

END $$
```

Jak można zauważyć procedura sprawdza najpierw czy kategoria i użytkownik istnieją w bazie danych a dopiero potem dodaje zawartość do bazy.

Przykładowe wywołanie

```
call db.createPost('test','testtesttesttesttesttesttesttest', 'Sprawozdania','af@af.com');
```

Oto jej wynik

idPost	name	content	enabled	creationDate	CategoryId	PosterId
103	test	testtesttesttesttesttesttesttest	1	2020-11-21	3	47

Przykład prostej funkcji zliczającego ilość ocen w komentarzu

```
drop procedure if exists `db`.`countVotesByCommentId`;
delimiter $$

CREATE PROCEDURE `db`.`countVotesByCommentId`(
    commentId INT
)
BEGIN

    declare upvotes INT;
    declare downvotes INT;
    declare votes INT;

    set upvotes = db.countVotes(commentId, 1);
    set downvotes = db.countVotes(commentId, 0);

    set votes = upvotes - downvotes;

    select commentId, votes;


END $$
```

Procedura wykorzystuje funkcje które zwracają ilość “łapek w górę i w dół” i je przelicza i wyświetla tą wartość wraz z id Komentarza

Przykład procedury dodającej nagrodę dla użytkownika

```
drop procedure if exists `db`.`rewardUser`;
delimiter $$

CREATE PROCEDURE `db`.`rewardUser`(
  givenRewardingUserId INT,
  givenMedalId INT,
  givenContent VARCHAR(100),
  givenPosterId INT
)
BEGIN

    IF EXISTS ( select * from db.user where db.user.idUser = givenPosterId ) then
        IF EXISTS ( select * from db.user where db.user.idUser = givenRewardingUserId ) then
            IF EXISTS ( select * from db.medal where db.medal.idMedal = givenMedalId ) then
                insert into db.reward(RewardingUserId, MedalId, content, crationDate, PosterId)
                value
                (givenRewardingUserId, givenMedalId, givenContent, curdate(), givenPosterId);
            END IF;
        END IF;
    END IF;

END $$
```

Jak można zauważyć procedura sprawdza czy encje istnieją w bazie danych i dopiero tworzy nowy, w przeciwnym razie nie wykona nic

dodatkowo moja baza danych pozwala na update dowolnych elementów, pusty string oznacza że dana nie będzie updatowana

```
drop procedure if exists `db`.`updatePost`;

delimiter $$

CREATE PROCEDURE `db`.`updatePost`(
    postId INT,
    givenName varchar(50),
    givenContent varchar(500),
    categoryId INT
    )
BEGIN

    if exists (select * from db.post where idPost = postId ) then
        if exists (select * from db.category where idCategory = categoryId ) then
            UPDATE post
            SET CategoryId = categoryId
            WHERE idPost = postId;
         end if;

            if (NULLIF(givenName, '')) is not null then
                 UPDATE post
                 SET name = givenName
                 WHERE idPost = postId;
            end if;

            if (NULLIF(givenContent, '')) is not null then
                 UPDATE post
                 SET content = givenContent
                 WHERE idPost = postId;

        end if;
    end if;
END $$
```

Ostatnią wyróżniającym się elementem jest funkcja która zlicza ilość ocen użytkowników (wywołanej w procedurze kilka stron wyżej)

```
drop function if exists `db`.`countVotes`;

delimiter $$
CREATE FUNCTION `db`.`countVotes` (
    commentId int,
    voteType BIT(1)
    )
RETURNS INTEGER
BEGIN

    declare votesCount INT;

    select count(*) into votesCount from db.vote where vote.rate = voteType and vote.CommentId = commentId;

RETURN votesCount;
END $$
```

Jak można zauważyć funkcja liczy wszystkie oceny w konkretnym komentarzu i o konkretnej wartości i zwraca ją potem jako INT

Prezentacja problemów w realizacji

Nie napotkałem większych problemów w realizacji, największym wyzwaniem było stworzenie generatora insertów do bazy danych. Dzięki nim w krótkim czasie udało dodać do bazy kilkadziesiąt tysięcy encji.

Link do repozytorium
https://github.com/rmagrys/StudentForumDb

3. Wnioski

Bazy danych to jedna z najważniejszych, jeśli nie najważniejszy elementów aplikacji, przechowuje bezpiecznie dane z których aplikacje klienckie korzystają. Bez bazy danych wszystkie informacje musiałby być przechowywane w pliku lib “in memory”, co bardzo komplikuje sprawę i nie daje nad dostatecznej ochrony informacji które przechowujemy