Bazy danych - dokumentacja

Bartosz Bednarczyk

Instytut Informatyki, Uniwersytet Wrocławski, Polska
bbednarczyk@stud.cs.uni.wroc.pl

1 Model konceptualny i fizyczny bazy danych

Przedstawiona przeze mnie baza danych składa się z dziesięciu tabel widocznych na diagramie.

- W tabeli "service" przechowujemy współrzędne geograficzne serwisów rowerowych wraz z ich adresem.
- W tabeli "station" znajdują się informacje na temat adresu stacji, liczby wolnych stanowisk na rowery, pojemność stacji (czyli ile rowerów możemy do niej dostawić), jak i również współrzędne geograficzne stacji (potrzebne by umieścić znacznik na mapie). Dodatkowo przechowujemy liczbę wolnych miejsc na rowery, która obliczana jest na nowo przy każdym zapytaniu do bazy danych.
- Tabela "usser" odpowiada za użytkowników naszego portalu. W tabeli przechowujemy imię oraz nazwisko użytkownika, jego adres, nr pesel (który będzie służył jako login), adres email, nr telefonu, stan konta oraz hash hasła użytkownika. Trzymamy również informację o aktualnie wypożyczonym przez użytkownika rowerze (lub null jeśli aktualnie nic nie wypożycza).
- Informacje o rowerach przechowujemy w tabeli "bike". Znajdują się tam takie informacje jak nazwa modelu, kolor, ostatnia data przeglądu oraz boolowska wartość oznaczająca czy rower jest w dobrym stanie. Dodatkowo każdy rower ma przypisaną stację (która być może jest nullem, kiedy rower został wypożyczony, ale jeszcze nie oddany).
- Tabla "serviceman" odpowiada wpisom na temat serwisantów. Każdy z nich ma przypisane imię i nazwisko (naprawdę?) oraz adres email i nr telefonu.
- W tabeli "issue" trzymane są wpisy na temat ewentualnych usterek rowerów. Każdy z użytkowników może zgłosić usterkę dotyczącą pewnego roweru wraz z jej opisem (być może pustym, jeśli użytkownik nie ma chęci dokładniej jej opisywać). Każda naprawiona usterka posiada finishdate, który nie jest nullem. Każda usterka posiada czas jej zgłoszenia.
- W tabeli rent trzymamy informacje na temat aktualnych oraz archiwalnych wypożyczeń z serwisu. Każde wypożyczenie dotyczy pewnego użytkownika oraz roweru. Posiada ono również czas wypożyczenia oraz zwrotu roweru.
- Tabela payment odpowiada wpłatom użytkowników. Zasilanie konta pozwala na wynajmowanie rowerów na dłuższy okres.

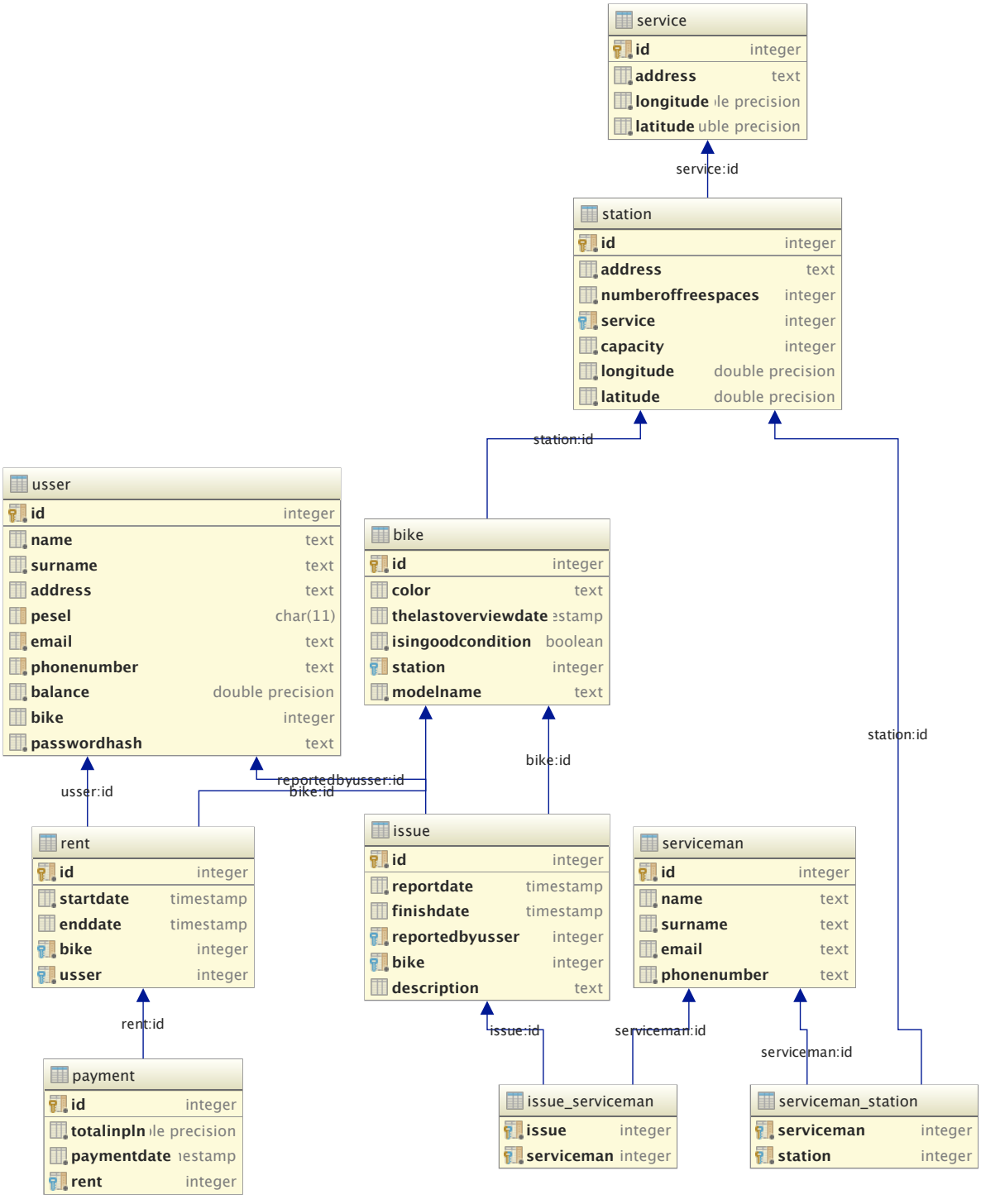
W bazie danych uwzgledniłem trzy role :

- administrator,
- serwisant oraz
- użytkownik

Admin ma oczywiscie dostep do kazdego kawałka bazy danych. Uzytkownik moze wypozyczac rower, zmieniac informacje o sobie, dodawac nowe płatnosci oraz uwagi na temat sprzetu. Serwisant ma dostep do serwisu, stacji oraz swoich danych i zgłoszen. Nieopisane fragmenty sa modyfikowane przez administratora.

Dokładny opis tabel wraz z poleceniami tworzącymi całą bazę można znaleźć w pliku "database_create_script.sql".


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


service
id integer
address text
longitude le precision
latitude uble precision
service:id
station
id integer
address text
numberoffreespaces integer
service integer
capacity integer
longitude double precision
latitude double precision
station:id
usser
id integer
name text
surname text
address text
pesel char(11)
email text
phonenumber text
balance double precision
bike integer
passwordhash text
bike
id integer
color text
thelastoverviewdate estamp
isingoodcondition boolean
station integer
modelname text
station:id
bike:id
reportedbyusser:id
bike:id
usser:id
rent
id integer
startdate timestamp
enddate timestamp
bike integer
usser integer
issue
id integer
reportdate timestamp
finishdate timestamp
reportedbyusser integer
bike integer
description text
serviceman
id integer
name text
surname text
email text
phonenumber text
rent:id
issue:id
serviceman:id
serviceman:id
payment
id integer
totalinpln le precision
paymentdate estamp
rent integer
issue_serviceman
issue integer
serviceman integer
serviceman_station
serviceman integer
station integer

2 Aplikacja

Aplikacja jest podzielona na dwie części - frontend oraz backend. Aplikacja została zaprojektowana w stylu RESTful, czyli zgodnie z dzisiejszymi trendami w informatyce.

W części backendowej skorzystałem ze Spring Framework (pol. zrębu wiosna), JDBC jako sterownik do bazy danych Postgresql, Jersey do realizacji zapytań restowych oraz Gradle jako system budowania projektu.

W części frontendowej korzystam z javascriptu - dokładniej AngularJS. Korzystam również z Bootstrap, HTML, SCSS. Do pobierania bibliotek do angulara stosowany jest npm. Do pobierania bibliotek związanych z bootstrapem używam bowera. Systemem budowania oraz serwerem jest gulp.

Kod w części backendowej podzielony jest na :

- dao (ang. data access object) - klasy obiektów w bazie danych
- daoimpl - implementacji powyższych klas i zapytań
- serwisy - klasy przekierowujące odpowiednie zapytania restowe do odpowiednich daoimpl oraz generuję odpowiedzi (ang. response)

Kod w części frontendowej podzielony jest na :

- index.html - główna strona aplikacji
- img - obrazki
- scss - arkusze stylów
- app.js - główne aplikacja po stronie javascriptu
- js controllers - część logiki biznesowej po stronie frontendu
- js directives - wydzielone fragmenty kodu HTMLowego opakowane w nowe znaczniki
- js services - serwisy po stronie frontendowej

3 Sposób uruchomienia aplikacji

Żeby uruchmić projekt:

- Instalujemy virtualbox i vagrant
- W katalogu projektu wpisujemy vagrant up
- Wpisujemy vagrant ssh
- Przechodzimy do katalogu /vagrant
- Uruchamiamy my_script.sh
- W katalogu projektu wpisujemy polecenie gradle clean build Backend:run, a w katalogu web polecenie gulp webserver:live
- Uruchamiamy ulubioną przeglądarkę internetową i wchodzimy na adres 10.100.100.150:8000

Polecenie vagrant up stawia nową maszynę wirtualną, na której instalowany jest system ubuntu wraz z zależnościami projektu (ponglish: dependencje).

Uwaga! Na maszynę wirtualną rezerwowane jest 1024MB pamięci RAM.