\subsection{Testy latencie}

\noindent Testy rýchlosti našej aplikácie sme vykonávali na našom serveri s Ubuntu 16.04.2 LTS S jedným virtuálnym jadrom a 4GB RAM. Testovali sme na databáze Neo4j Community verzie 3.1.4 a na databáze MYSQL verzie 5.7.18, tabuľky boli uložené pomocou InnoDB.

Ako dátovú sadu sme použili reálne dáta o destináciách a trasách z Rome2Rio API. Aby sme zabezpečili zvyšovanie ako veľkosti tak aj hustoty grafu začali sme s jedným používateľom s desiatimi položkami, v druhom kroku sme pridali používateľa s päťdesiatimi destináciami, ktorý mal zároveň pridané všetky destinácie prvého používateľa, takto sme pokračovali až po počet destinácií 1400. Dáta sme generovali našou aplikáciou a následne migrovali do MYSQL databázy štruktúrovanou ako v našom návrhu \ref{database-diagram}

Prvý test, ktorý sme robili bol syntetický test rýchlosti komplexného dopytu na databázu. Cieľom testu bolo porovnať škálovateľnosť relačnej a grafovej databázy v prípadoch podobných nášmu. Dopyt ktorého latenciu sme testovali bol na zoznam obľúbených destinácií, ku každej destinácií sme vyhľadali ešte trasu ktorá k nej vedie z domáceho miesta používateľa a zoznam ostatných používateľov ktorý tiež majú túto destináciu na zozname. Dopyt v databáze Neo4J môžeme vidieť na príklade \ref{cypher-follows}. Dopyt v databáze MYSQL môžeme vidieť na príklade \ref{sql-follows}.

\begin{algorithm}

\lstset{

language=SQL,

basicstyle=\small\sffamily,

frame=none,

numbers=left,

xleftmargin=5.0ex,

numberstyle=\tiny,

stepnumber=1,

showstringspaces=false,

keywordstyle=\color{blue}\bfseries

}

\lstset{emph={% Adjust any special keywords

printf%

},emphstyle={\color[rgb]{1,0,0}\bfseries}%

}%

\begin{lstlisting}

SELECT \* FROM `follows` f

LEFT JOIN `home` h ON h.`id\_user` = f.`id\_user`

LEFT JOIN `place` p ON f.`id\_place` = p.`id`

LEFT JOIN `routes` r ON r.`id\_from` = h.`id\_place` AND r.`id\_to` = p.`id`

LEFT JOIN `follows` fo ON fo.`id\_place` = p.`id`

LEFT JOIN `users` u ON u.id = fo.`id\_user`

LEFT JOIN `tsp` t ON t.`id\_user` = f.`id\_user` AND t.`id\_place` = f.`id\_place`

WHERE f.`id\_user` = 231

\end{lstlisting}

\caption{SQL dotaz pre testovanie rýchlosti}

\label{sql-follows}

\end{algorithm}

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|l|l|l|l|l|l|l|l|}

\hline

Počet položiek& 10 & 50 & 100 & 200 & 500 & 1000 & 1400 \\ \hline

Cypher shell & 6,7 & 25,2 & 32 & 54 & 99,8 & 162,7 & 248,9 \\ \hline

SQL & 15,5 & 16,1 & 31,6 & 62,5 & 387,3 & 1339,8 & 2396,6 \\ \hline

\end{tabular}

\end{table}

\begin{figure}

\centering

\includegraphics[width=14cm]{img/speed-sql.pdf}

\caption{Výsledky testu rýchlosti API pre zobrazenie zoznamu obľúbených destinácií}

\label{speed-plot-sql}

\end{figure}

Druhý test ktorý sme robili bol reálny test latencie nami implementovaného API v dvoch verziách. V prvej verzii s použitím OGM a v druhej s použitím PHP Neo4J Client. Tento test sme vykonali ako súčasť implementácie, aby sme vedeli posúdiť ktoré riešenie použijeme vo finálnej verzií aplikácie.

Pri teste sme používali rovnakú dátovú sadu ako v prvom teste. Latenciu sme merali tak, že sme posielali dopyty na naše API a zaznamenávali sme čas od doručenia dopytu na server do začiatku sťahovania dát. Na údajoch v tabuľke č. \ref{speed-table} a grafe č. \ref{speed-table} môžeme vidieť ako latencia nášho API narastala s počtom záznamov na zozname obľúbených.

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|l|l|l|l|l|}

\hline

Počet položiek & 10 & 50 & 100 & 200 \\ \hline

NeoEloquent & 101,6 & 281,7 & 705,7 & 1857 \\ \hline

PHP neo4j client & 83,2 & 195 & 358,5 & 695,7 \\ \hline

\end{tabular}

\end{table}

\begin{figure}

\centering

\includegraphics[width=14cm]{img/speed.pdf}

\caption{Výsledky testu rýchlosti API pre zobrazenie zoznamu obľúbených destinácií}

\label{speed-plot}

\end{figure}

Ďalší udaj je latencia manuálne spusteného dopytu rovnakého ako v druhom prípade ale cez cypher-shell. Výsledky testovania latencie cypher-shell zahŕňa len čas ktorý trvalo vykonanie dopytu (informáciu poskytuje cypher-shell), zatiaľ čo ostatné údaje zahŕňali aj latenciu celej našej aplikácie ako: jednorazových operácií ako beh APACHE, PHP a štart frameworku Laravel, tiež zahŕňali latenciu prenosu dát medzi Neo4J a PHP, konverzie JSON dokumentov na PHP objekty a spravovanie údajov v príslušných knižniciach.