Henry Fang, Liam Neufeld
hef052, lwn282
11233914, 11232603
CMPT 353

Part A

The initial application we built was based on the MySQL database <-> NodeJS server <-> HTML client structure that we are familiar with from the assignments. A client who talks with a server and the server then talks to the database based on the needs of the client. Basically, an html client and a MySQL database interacting through the middleman that is the NodeJS server.

The database is structured with a table for staff, for customers, and for reports. The staff and customer table store their own information as well as a unique integer ID. The reports feature the id of the customer they are attached to, their own id, and the report itself.

The project was built along the idea of an administrator interface for a Community Based Organization. The type of CBO we imagined was one that sought to help lower-income families. Not necessarily homeless people, but rather people that needed a little material boost on occasion. We have all the mentioned functions for registering, deleting, editing, ang viewing staff and customers. We also have report adding for customers for any comments that staff could make such as checkup visits by staff, or they came in for some reason such as they needed help with health expenses etc.

For functionality testing we simply went in and used the system. Performing all the various actions on staff and customers, and seeing if expected updates are performed. Also ran some load tests. Although this type of web application probably wouldn’t see more than a few requests simultaneously we still wanted to see how it would fair. For registering both staff and customers the speed was quite good with latencies of less than 5ms for 100 requests.

Henry Fang, Liam Neufeld
hef052, lwn282
11233914, 11232603
CMPT 353

Part B

We chose MongoDB as a technology since it is one of the most popular database technologies. Used by big names like Toyota, Adobe, and even Google. It seemed like a solid technology that will be around for many years. It is also a NoSQL database which differentiates it from MySQL which we used in Part A. The JSON document structure of the data also meant that is had some similarities to CouchDB form which we already had some experience to draw upon.

We also chose to use their cloud database service Atlas. After initially experimenting with setting up a docker containerized MongoDB database it turned out to be much more convenient to use Atlas. Using Atlas meant that we didn't have to worry about initializing anything through docker. No dealing with ports, volumes, or any unexpected quirks that docker can sometimes have.

Part C

MongoDB is classified as a NoSQL database type and thus it has no need for any knowledge of SQL to use. The structure of the database includes databases themselves, Collections, and Documents. The Collections would be most similar to the tables that exist in SQL databases and the Documents like the rows within the tables. Each Document resides within a Collection as a JSON-like object.

The Collections are created from Schemas which are JSON objects that define the structure and contents of a Collection. Similar to defining the columns of an SQL database. However, unlike SQL, MongoDB does not require Documents in the same Collection to have the same Schema.

Henry Fang, Liam Neufeld
hef052, lwn282
11233914, 11232603
CMPT 353

While SQL tables are rigid in the available fields and the datatypes of those fields, MongoDB allows whatever datatypes and number of new fields you want to add for a particular Document. There are options to restrict this as in most cases having the same structure within Documents is beneficial. However, the point is that you have the option to do more.

In terms of accessing the data in our NodeJS server, we used the Mongoose library. Rather than constructing query strings like SQL there are methods which are used for all the operations.

In terms of usage cases, it offers a well-made alternative to the classic SQL databases. It can handle all the features and ideas of SQL, but also offers additional flexibility that SQL restricts. It also seems like it was designed a lot more with web programming in mind by storing data as JSON-like objects, allowing quick manipulation in JS based systems.

Part D

The new application design is very similar to Part A. Instead of the MySQL database we have the server communicate with our new MongoDB Atlas database. Implementing this simply meant redoing the NodeJS server functions to connect to and access our new DB. So, our new architecture is MongoDB Atlas Database <-> NodeJS <-> HTML client.

We used Mongoose (The NodeJS MongoDB library) to connect the functions in our server. All the function routes remained the same so that the client didn’t need any changes, only the function bodies had to access our new database. Mongoose was incredibly nice to use. Instead of having to construct various SQL query strings for each operation Mongoose instead has nice methods for each operation we needed. This was quite nice as syntax errors were caught immediately by the IDE instead of when the database received a bad request in SQL.

The cloud-based database was also a massive improvement over running out database in docker. Initially we did try to run MongoDB in a docker container, but upon discovering Atlas we quickly switched. With Atlas we didn't have to worry about running another server and having to deal with ports, volumes, and all the other little details. All we had to do was set up an account then simply copy the access string into our NodeJS server to connect. Another feature was that this database was persistent. We didn't have to reinitialize it every time or save a dump of the data.

For testing functionality, it was much like Part A where we just clicked through all the buttons and looked for correct updates to the system. We also ran some load tests. In terms of performance is suffered a severe decrease of around 120ms. This is to be expected since we have the extra communication to the cloud server while in Part A we were connected to a locally hosted server. If the MySQL server was hosted on a cloud server, I would expect the results to be similar.