DATABASE RAPPORT

Hassan Raza Hussain, Hassan Mohdi, Gordan Mikkelsen & Kristoffer Rath hansen

29. MAJ 2020
CPHBUSINESS
GITHUB LINK: https://github.com/lakridserne/db-exam

INDHOLD

..

INTRODUCTION

We are a group of 4 members who have been given an assignment for the database exam project. The objective of this assignment is to provide database support to an application, as a proof of concept. In the assignment, multiple diverse database models are built and implemented for different purposes.

We have chosen to deal with data from two sources, which has to do with houses and apartments in Denmark. One source, DAWA, contains a list on every address in Denmark and additional information regarding those addresses that is public to the general public. The other source is from a real estate agent containing data about houses and apartments sold in Denmark.

Based on these data sources, we have made the following problem statement.

PROBLEM STATEMENT

Getting a quote on your house or apartment can take time. If you want to quickly find out roughly where you stand can be very challenging. Therefore we want to give an immediate rough estimate on your house or apartment based on available public data

DESIGN AND ARCHITECTURE

Requirements and analysis

Business case

Contribution to the business strategy	Our strategy is to provide homeowners with a tool to quickly estimate the value of their house or apartment. This is done through analysis of the public data regarding the address and the area.
Options considered	1. More and better advertising to get more customers to find out how much value their house or apartment has. 2. Live with slow quotation time and thus less customers buying and selling 3. The new system
Benefits	1. Immediate quotes 2. Increased sales based on the immediate quotes
Timescales	Our current analysis shows us that the system will take around 6 weeks to develop
Cost	The new system is estimated to cost around $20,000.

Expected return on investments	It is expected that the system will recoup the costs and start giving a return on investment through increased sales due to immediate quotes.
Risks	There are various risks such as inaccurate quotes that we are trying to mitigate through sampling different house prices in different areas and compare it with real listings.

User stories

User story 1 - House price

As customer on the website, I want to find out what my house is worth, so I go to our website where I enter my house or apartments address. Then the website will respond with an approximate price with how much the house is worth. That way, I can better decide if I want to sell my house.

User story 2 - number of rooms on a specific house

As customer on the website, I want to find out how many rooms a specific house has that was sold by Boliga in the past. Therefore, I go to our website, where I enter address on the specific house, the website then respond back with the number of rooms in the house.

So I can see if the specific house fits my needs

User story 3 - Graphs

As salesman on the on database, I want to see statistics and graphs on the different houses and apartments in question.

This way, I can better target the right customers.

Database fields

Our Database fields is designed on the way to give you access to accurate the information you want, since a proper design on our database is essential for reaching our goals, even though we spent a lots of time on it, it is worth it since we end up with a database that meets our needs and can easily be adapted to changes. In our database We don’t really have a lots of different fields, because we are mostly working with basic data, where most of our data type are either varchar or int since we feel like they fits best with data we have in our boliga database.

Database structure

In the following, we’ll discuss how we designed our database structure and chose which technologies were appropriate to use taking the user stories and business case into consideration.

Design of databases

We have chosen 4 different databases to work with (per the requirements of the assignment). The databases were selected based on criterias we’ll delve deeper into below, but first, I’d like to go through what data we have and how we are planning on using those data.

In our business case, we have some data from sales of houses and apartments. Based on these sales, and based on public data from DAWA (Danmarks Adresse Web API) we will be giving some very rough estimates on how much you could get for selling your house or apartment in your area.

In order to fulfill that we need some databases to store our data in. We have thus decided to have databases with these purposes.

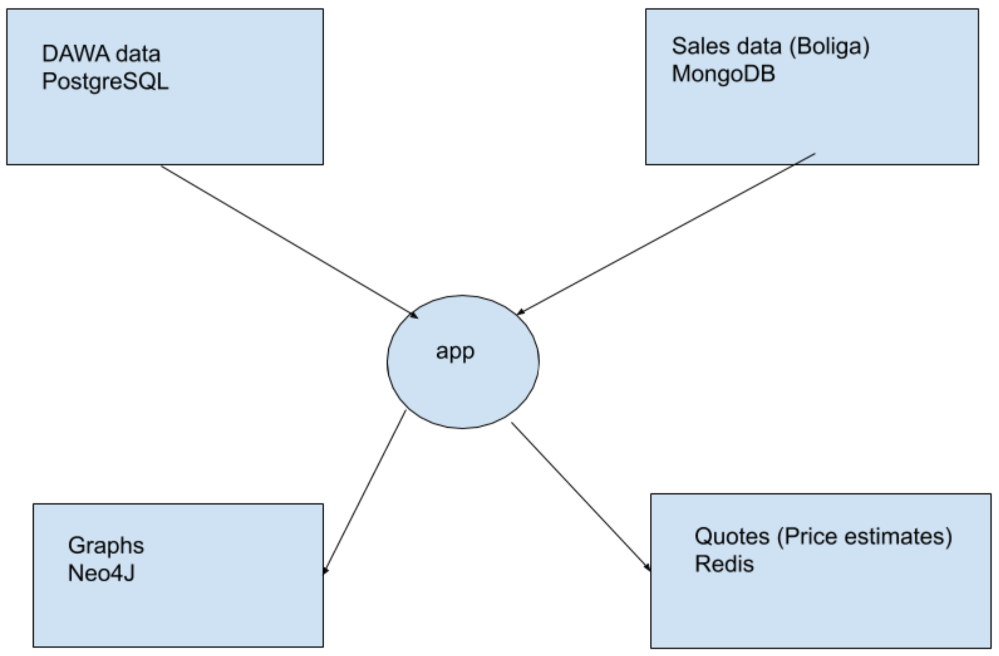
- Database for DAWA data (general information about addresses in Denmark)
- Database for boliga data (data showing which sales boliga have made)

- Database used for generating graphs for statistics purposes
- Database for storing the data we generate

Could we have stored the data in the same database? Yes, we could theoretically have stored a large part of the data in the same database if we wanted to. However, if we do that, the underlying databases will not be optimized for the data we store and process.

By using these 4 different databases, we can choose each one from what things we expect to store, how popular we expect it to, how resilient it should be, and many more parameters.

Below you can see a diagram detailing what databases we will be using for what data. We'll delve deeper into the choices of the different databases in "Choice of database technologies".



(IMAGE 1)

Choice of database technologies

We have evaluated the different databases on a number of criteria to see which database types would be good to use for our data.

One of the things we thought about when deciding which databases we wanted to use was whether we should use relational databases or NoSQL databases. Relational databases are pretty good if you have data where you need to relate to it with some other specific data.

However, NoSQL databases can be faster than relational databases in certain circumstances (note not in all) and allows you more flexibility to store different kinds of data (where relational databases are more strict). This can be good because it allows you to stretch your models a bit, but it can also be dangerous if you do it too much because you can end up having inconsistent data or data structured in a way so it'll take a very long time to find the data you are looking for.

We chose to have our data from DAWA in a PostgreSQL as that is a relational database. This is good for the kind of data we have in DAWA, where we want to quickly be able to find specific addresses from different informations.

For our sales data, we have chosen MongoDB to store those. The reason we chose to store our sales data in a NoSQL database is that it is highly scalable (you can just launch more mongod processes).

We have a lot less data to search through with the sales data and can with indexes optimize things a lot so performance is optimal.

Neo4J was without a question the choice for making graphs for us, since it's made for making graphs primarily.

Lastly, we have Redis that are driving our quotes database. Redis is like MongoDB (and for a part also PostgreSQL) a NoSQL database. The advantage of using Redis is that it's very fast as it runs in-memory.

We also looked at things like CAP and ACID for the different databases.

	CAP qualities	ACID
MongoDB	MongoDB is highly available	From MongoDB 4.0 multi-

	through automatic failover and strongly consistent by default. Therefore the CAP theorem is fulfilled.	document ACID transactions is possible.
Redis	Since Redis is not designed to be run on several servers, it is not as highly available as the other databases. However, it is consistent and tolerates partitions. Therefore it follows CP and not CAP.	Redis is single threaded allowing it to be ACID compliant by default.
Neo4J	Neo4J only follows CA and is not partition tolerant.	Neo4J is fully ACID compliant.
PostgreSQL	PostgreSQL is not partition tolerant - so it’s CA.	PostgreSQL is ACID compliant from the ground up.

Implementation of databases

Logging

When we build applications and services, it's being more and more important to be able to discover information about the running system. More precisely being able to analyse the information to diagnose problems, discover information so we can avoid problems before they become severe, or just to discover information about users. All this requires logging.

Redis:

Redis has built in logging capabilities. In the config file you can set up where it should send the log lines to - for example to stdout (the command prompt) - or a specific file. In Linux it also have native support for syslog. Syslog accepts log messages from every program that sends it a message and it routes those message to various on-disk log files, handling rotation and deletion of old logs. With configuration, it can also forward messages to other server for

further processing. When we talk about it as a service, it’s much more convenient than logging to files directly, because things like all of the special log file rotation and deflection is handled for us already.
PostgreSQL

By having PostgreSQL run like a cloud service at Amazon AWS, it gives us an opportunity to see the logging on Amazon's website. It gave us a overview of which events there has been in our PostgreSQL database. In the overview for logging it shows the time for the recent event in the database and a system note for what the event was.

MongoDB:
Before we could enable logging for MongoDB, we had to set a single verbosity for log messages. That had the following form:db.setLogLevel(<level>, <component>).

In this form, <level> take int as its type, and <component> take string as its type. db.setLogLevel() sets a single verbosity level. If we had to set more than one verbosity level in a single operation, we had to use a setParameter command to set the LogCommponentVerbosity parameter. Another way we could do it was to specify verbosity settings in the configuration file.

Neo4J:

Neo4J does contain some possibilities for logging. Thus it does save some log information in some log files. However, most of the log functions is only for the enterprise version of Neo4J, and it was not that relevant for our project.

Session

Redis:

To store the sessions in Redis what we do is that we use Redis hashes which is a pretty standard way for session state. “Redis Hashes are maps between string fields and string values" ([1]) . So we use the session ID as the key and to use the hash fields to describe the state. ([2])

MongoDB:

Regarding sessions storage we implemented an API called Spring Session MongoDB. This allows us to manage a users session information and storing it in MongoDB. The dependencies is added in the xml file so we can import the framework. We do this because we want to get an overview of interaction with the application in case something goes wrong. ([3])

PostgreSQL:

When we store sessions in PostgreSQL, we gather the database information that is important. After that we create a session store schema. Then follow a tutorial on how to point the policy server to the database. Lastly, we restart the server so the changes goes into effect ([4])

Transactions

Redis:

Redis is different compared to most of the other database transactions, where it can use MULTI, EXEC, DISCARD and WATCH as the foundation of transactions. Redis also allows you to execute a group of command on a single step.

Compared to our other databases, Redis does not support roll backs, which means it can fail during a transaction, where it will keep trying to execute the rest of the transaction.

[1] https://redis.io/topics/data-types

[2] https://stackoverflow.com/questions/54798523/how-to-configure-redis-cache-for-session-management-with-uuids

[3] https://spring.io/projects/spring-session-data-mongodb#learn

[4] https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/ca-enterprise-software/layer7-identity-and-access-management/single-sign-on/12-7/installing/install-a-policy-server/configure-odbc-databases-as-policy-session-key-and-audit-stores/configure-odbc-databases-as-session-store/store-session-information-in-postgresql.html

PostgreSQL;

PostgreSQL has been having transaction support for long time, where you have to use the BEGIN, COMMIT or ROLLBACK queries through a client to control the transaction, because node-postgres is a low level and un-opinionated language. PostgreSQL has support for ACID per default.

MongoDB:

After Mongodb updated to version 4.0, they now have support for multi-document ACID transactions, and bit later in version 4.2 they also added support for distributed transaction, which make you able to do multi-document transaction on a sharded clusters and incorporates the existing support for multi-document transaction on replica sets.

Installation instructions

Mongo:

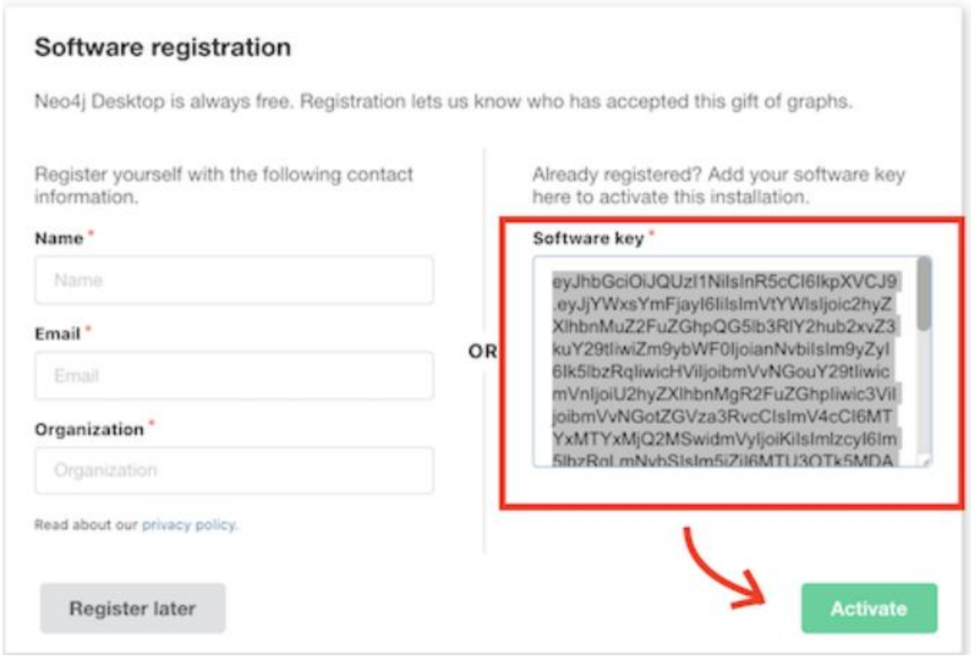
We installed MongoDB community Edition on Windows. We used the official MongoDB documentation to install it. We installed MongoDB version 4.2. Before we could install MongoDB and work with the database we had to be sure that mongoDB 4.2 community edition supports our version of Windows, and since we met the requirements, we could use MongoDB without any problems. Users that use earlier Windows versions than Windows 10 must install an update before installing mongoDB.[5]

Neo4J:

When we were installing Neo4J, the first thing we did was, to download it from their website along with going to the official documentation to get installation instructions. It was pretty easy work. Before we could start the Neo4J download on our computers we had to fill out some information about us on Neo4J's website. Then we could install the Neo4J desktop on our computers. The important thing here was we had to get the "Activation key", which you

[5] https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/

can see after we filled out the information as mentioned earlier. Another thing you also can do is if you not get the activation key for a reason, then you can also generate a key from within the app filling out the form on the right side of the app screen. [6]



Software registration

Neo4j Desktop is always free. Registration lets us know who has accepted this gift of graphs.

Register yourself with the following contact information.

Name *

Email *

Organization *

Read about our privacy policy.

OR

Already registered? Add your software key here to activate this installation.

Software key *

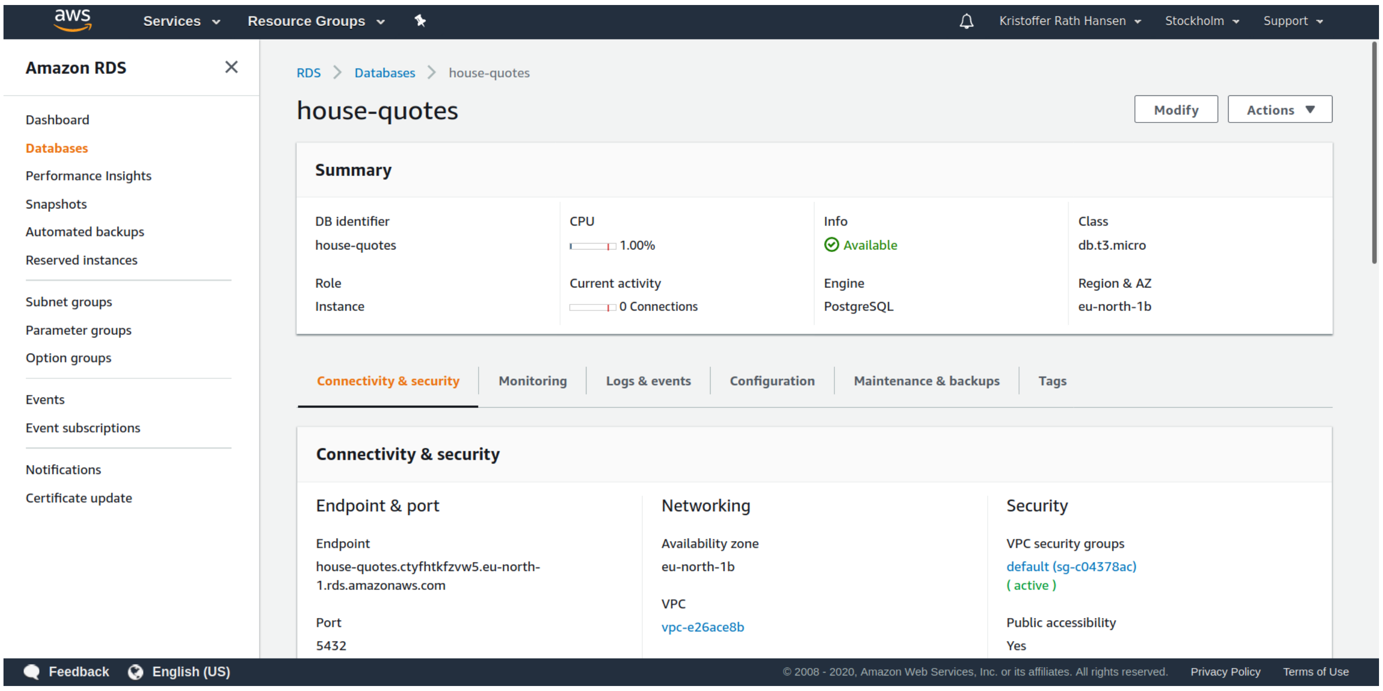
```
eyJhbGciOiJQUzI1NiIsInR5cCI6IkpXVCJ9
.eyJjYWxsYmFjayI6IiIsImVtYWlsIjoic2hyZ
XIhbnMuZ2FuZGhpQG5lb3RlY2hub2xvZ3
kuY29tIiwiZm9ybWF0IjoianNvbiIsIm9yZyI
6Ik5lbzRqIiwicHViIjoibmVvNGouY29tIiwic
mVnIjoiU2hyZXIhbnMgR2FuZGhpIiwic3ViI
joibmVvNGotZGVza3RvcCIsImV4cCI6MT
YxMTYxMjQ2MSwidmVyIjoiKiIsImlzcyI6Im
5lbzRqLmNvbSIsIm5iZiI6MTU3OTk5MDA
```

(IMAGE 2)

PostgreSQL:

In the start we downloaded and ran PostgreSQL on the computer. But afterwards we decided to run it like a cloud service on Amazon AWS. The reason we did that was our DAWA data was too big to run it on PostgreSQL on our local machine. We decided to run it on Amazon, because one group member knew we could do it that way. We experienced that doing it in that way it gave us a lot features and we don't needed to think about if the server is running. As we can see on the picture (IMAGE 3) below it shows us how our PostgreSQL dashboard looked like in Amazon.

6 https://neo4j.com/download-thanks-desktop/?edition=desktop&flavour=windows&release=1.2.8&offline=true#installation-guide



house-quotes

Summary

DB identifier	CPU	Info	Class
house-quotes	1.00%	Available	db.t3.micro
Role	Current activity	Engine	Region & AZ
Instance	0 Connections	PostgreSQL	eu-north-1b

Connectivity & security

Endpoint & port

Endpoint
house-quotes.ctyfhtkfzvw5.eu-north-1.rds.amazonaws.com

Port
5432

Networking

Availability zone
eu-north-1b

VPC
vpc-e26ace8b

Security

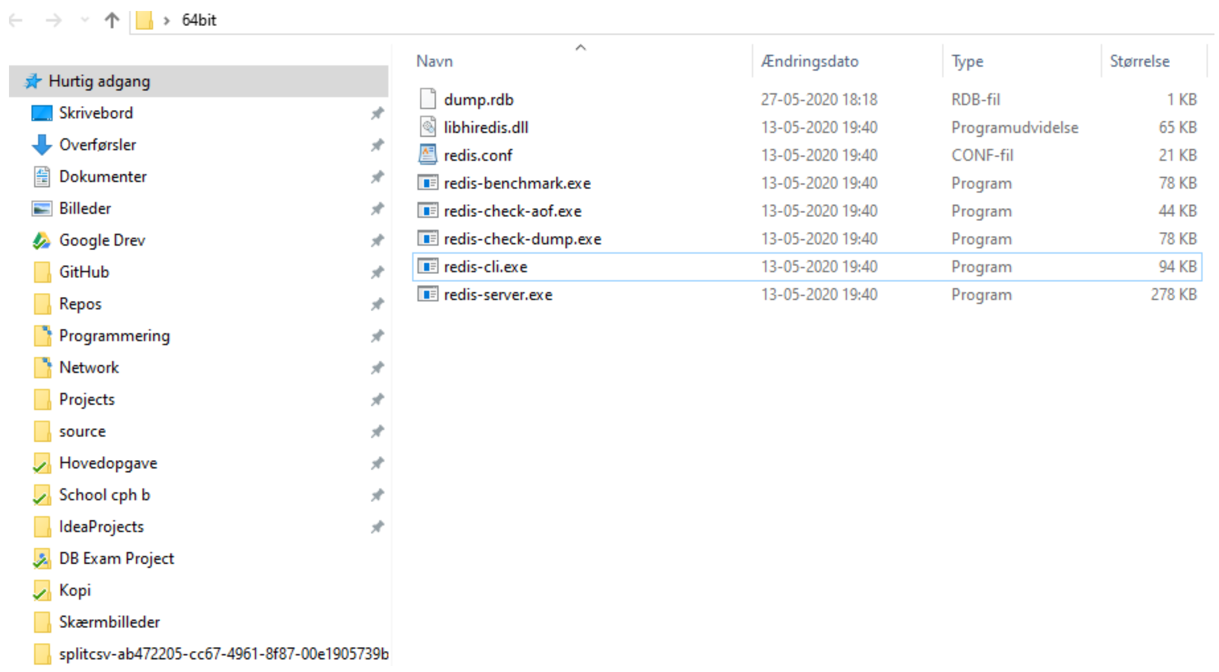
VPC security groups
default (sg-c04378ac)
(active)

Public accessibility
Yes

(IMAGE 3)

Redis:

When we installed Redis, we downloaded a zip folder which we unpacked. This folder contained a Redis server and Redis CLI. Then if we wanted to to start the redis server we just have to click on redis-server.exe. If we wanted to be sure about if the Redis server is running, we could check it by click on redis-cli.exe which also is a command prompt and in that we can write “ping” and then if the server is running it’ll answer with “pong”. In the image which is showed below (IMAGE 4), we can see how the Redis file looked.



64bit

- Hurtig adgang
- Skrivebord
- Overførsler
- Dokumenter
- Billeder
- Google Drev
- GitHub
- Repos
- Programmering
- Network
- Projects
- source
- Hovedopgave
- School cph b
- IdeaProjects
- DB Exam Project
- Kopi
- Skærmbilleder
- splitcsv-ab472205-cc67-4961-8f87-00e1905739b

Navn	Ændringsdato	Type	Størrelse
dump.rdb	27-05-2020 18:18	RDB-fil	1 KB
libhiredis.dll	13-05-2020 19:40	Programudvidelse	65 KB
redis.conf	13-05-2020 19:40	CONF-fil	21 KB
redis-benchmark.exe	13-05-2020 19:40	Program	78 KB
redis-check-aof.exe	13-05-2020 19:40	Program	44 KB
redis-check-dump.exe	13-05-2020 19:40	Program	78 KB
redis-cli.exe	13-05-2020 19:40	Program	94 KB
redis-server.exe	13-05-2020 19:40	Program	278 KB

(IMAGE 4)

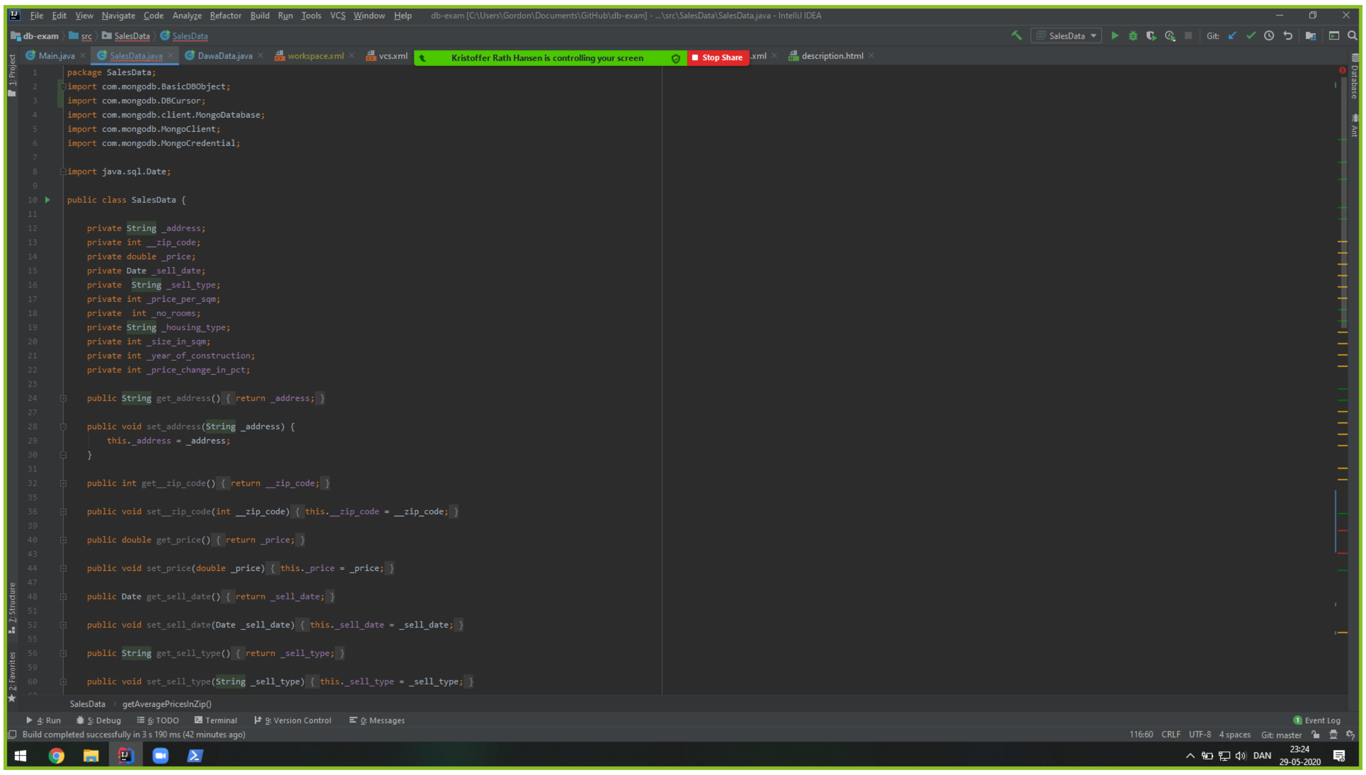
CONCLUSION

Based on our analysis and design we chose to use 4 databases, namely Redis, MongoDB, Neo4J and PostgreSQL and have them connectected to a single application. For creating the application , we did some research and chose Java since it seems to have a large user community for the databases we chose. After having connected the different databases to our application, we imported our two data sources, the first one into PostgreSQL and the other MongoDB.

Then by retrieving the different data from multiple sources the system will calculate an average price. These prices that gets generated will then get stored in our Redis database for future use.

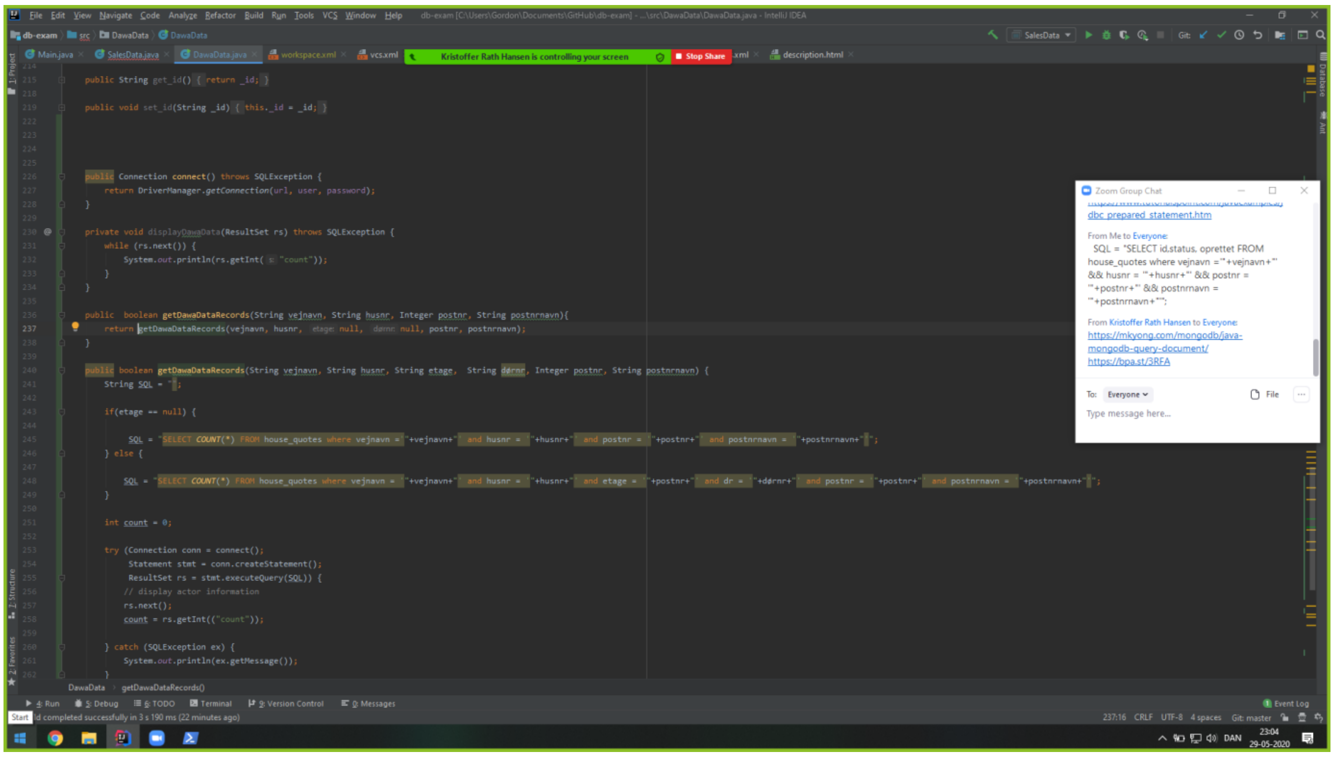
What our last database will do then is that Neo4j will collect the data that is necessary from MongoDB and PostgreSQL. From these data Neo4J will generate a graph which will display statistics.

BILAG:



```java
package SalesData;
import com.mongodb.BasicDBObject;
import com.mongodb.DBCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

import java.sql.Date;

public class SalesData {

    private String _address;
    private int __zip_code;
    private double _price;
    private Date _sell_date;
    private  String _sell_type;
    private int _price_per_sqm;
    private  int _no_rooms;
    private String _housing_type;
    private int _size_in_sqm;
    private int _year_of_construction;
    private int _price_change_in_pct;

    public String get_address() { return _address; }

    public void set_address(String _address) {
        this._address = _address;
    }

    public int get__zip_code() { return __zip_code; }

    public void set__zip_code(int __zip_code) { this.__zip_code = __zip_code; }

    public double get_price() { return _price; }

    public void set_price(double _price) { this._price = _price; }

    public Date get_sell_date() { return _sell_date; }

    public void set_sell_date(Date _sell_date) { this._sell_date = _sell_date; }

    public String get_sell_type() { return _sell_type; }

    public void set_sell_type(String _sell_type) { this._sell_type = _sell_type; }
```

Build completed successfully in 3 s 190 ms (42 minutes ago)



```java
    public String get_id() { return _id; }

    public void set_id(String _id) { this._id = _id; }




    public Connection connect() throws SQLException {
        return DriverManager.getConnection(url, user, password);
    }

    private void displayDawaData(ResultSet rs) throws SQLException {
        while (rs.next()) {
            System.out.println(rs.getInt( s: "count"));
        }
    }

    public  boolean getDawaDataRecords(String vejnavn, String husnr, Integer postnr, String postnrnavn){
        return getDawaDataRecords(vejnavn, husnr,  etage: null,  dørnr: null, postnr, postnrnavn);
    }

    public boolean getDawaDataRecords(String vejnavn, String husnr, String etage,  String dørnr, Integer postnr, String postnrnavn) {
        String SQL = "";

        if(etage == null) {

             SQL = "SELECT COUNT(*) FROM house_quotes where vejnavn = '"+vejnavn+"' and husnr = '"+husnr+"' and postnr = '"+postnr+"' and postnrnavn = '"+postnrnavn+"'";
        } else {

            SQL = "SELECT COUNT(*) FROM house_quotes where vejnavn = '"+vejnavn+"' and husnr = '"+husnr+"' and etage = '"+postnr+"' and dr = '"+dørnr+"' and postnr = '"+postnr+"' and postnrnavn = '"+postnrnavn+"'";
        }

        int count = 0;

        try (Connection conn = connect();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(SQL)) {
            // display actor information
            rs.next();
            count = rs.getInt(("count"));

        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
```

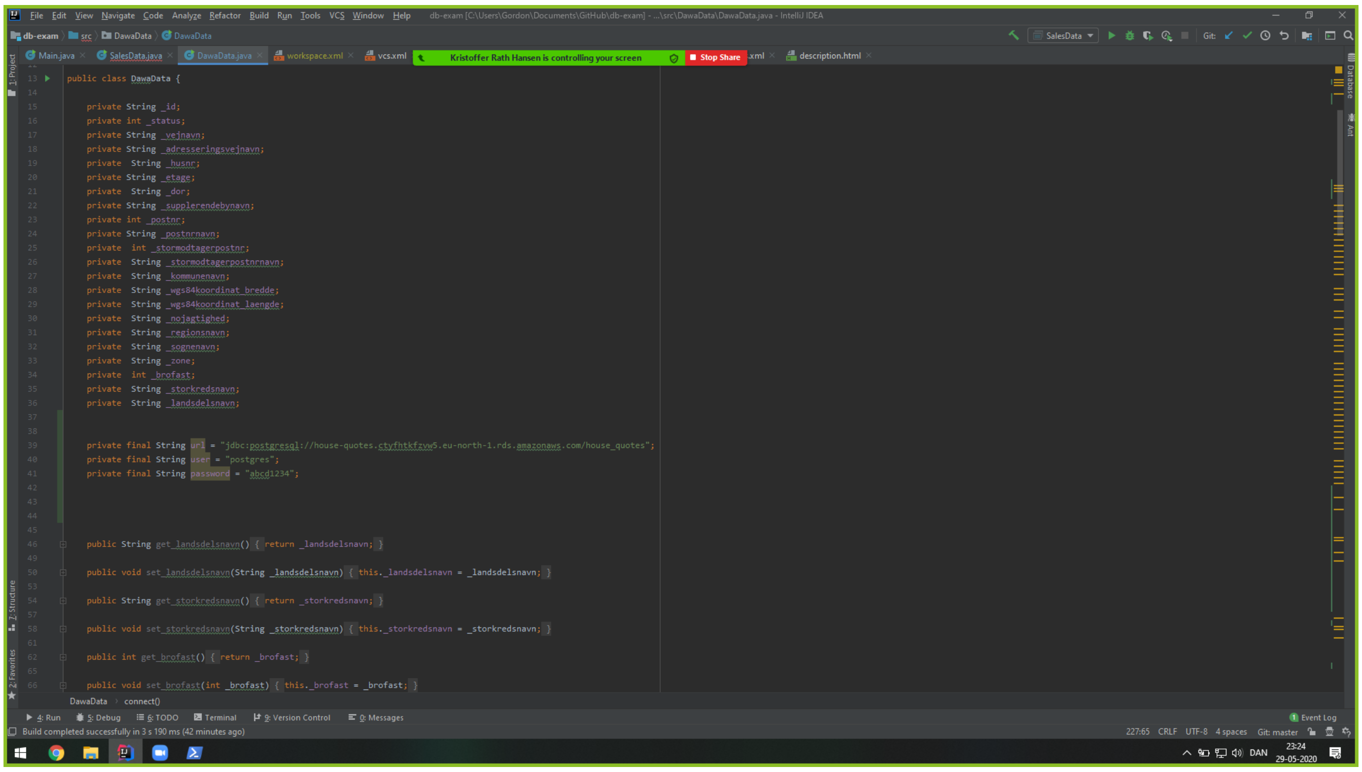
Zoom Group Chat

dbc_prepared_statement.htm

From Me to Everyone:
SQL = "SELECT id.status, oprettet FROM house_quotes where vejnavn ='"+vejnavn+"' && husnr = '"+husnr+"' && postnr = '"+postnr+"' && postnrnavn = '"+postnrnavn+"'";

From Kristoffer Rath Hansen to Everyone:
https://mkyong.com/mongodb/java-mongodb-query-document/
https://bpa.st/3RFA

ld completed successfully in 3 s 190 ms (22 minutes ago)



```java
public class DawaData {

    private String _id;
    private int _status;
    private String _vejnavn;
    private String _adresseringsvejnavn;
    private  String _husnr;
    private String _etage;
    private  String _dor;
    private String _supplerendebynavn;
    private int _postnr;
    private String _postnrnavn;
    private  int _stormodtagerpostnr;
    private  String _stormodtagerpostnrnavn;
    private  String _kommunenavn;
    private  String _wgs84koordinat_bredde;
    private  String _wgs84koordinat_laengde;
    private  String _nojagtighed;
    private  String _regionsnavn;
    private  String _sognenavn;
    private  String _zone;
    private  int _brofast;
    private  String _storkredsnavn;
    private  String _landsdelsnavn;


    private final String url = "jdbc:postgresql://house-quotes.ctyfhtkfzvw5.eu-north-1.rds.amazonaws.com/house_quotes";
    private final String user = "postgres";
    private final String password = "abcd1234";




    public String get_landsdelsnavn() { return _landsdelsnavn; }

    public void set_landsdelsnavn(String _landsdelsnavn) { this._landsdelsnavn = _landsdelsnavn; }

    public String get_storkredsnavn() { return _storkredsnavn; }

    public void set_storkredsnavn(String _storkredsnavn) { this._storkredsnavn = _storkredsnavn; }

    public int get_brofast() { return _brofast; }

    public void set_brofast(int _brofast) { this._brofast = _brofast; }
```

Build completed successfully in 3 s 190 ms (42 minutes ago)

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
db-exam [C:\Users\Gordon\Documents\GitHub\db-exam] - ...\src\DawaData\DawaData.java - IntelliJ IDEA
db-exam › src › DawaData › DawaData
Main.java
SalesData.java
DawaData.java
workspace.xml
vcs.xml
Kristoffer Rath Hansen is controlling your screen
Stop Share
description.html
SalesData
DawaData › connect()
4: Run
5: Debug
6: TODO
Terminal
9: Version Control
0: Messages
Event Log
227:65 CRLF UTF-8 4 spaces Git: master
DAN
23:24
29-05-2020