# Intro

We are a group of two, Christian Lind and Alex Langhoff. We have been tasked to create a system, built around the usage of up to 4 databases, to store the data used.

For this project, we have to write a rapport about the system and submit it as a part of our exam.

The github link to our repository for this report, is going to be at:
https://github.com/SOFT2020-2021/Database-recipes

We have chosen to make a forum, for people to share and find ideas to different recipes. You don't have to be a user to browse the forum of recipes, nor to browse the comments made to each recipe.

However, you do need to be a user on the site if you want to post your own recipe or add comments to another user's recipe or comment.
Hence it's a forum, the service of the site will be online, giving the person an overview to browse for ideas or ask for pointers to a certain recipe they want to try out.

# Statement of intent

This chapter will go in depth with the demands we have for our project.

The very first thing we had to start, was to build up an infrastructure to support our application, which had to be built around several different databases, which had different kinds of purposes in this project.

As stated earlier, this project is going to be built around the usage of up to 4 databases, or more if we felt the need to go further. Although, the more databases added to the project, the complexity of the program is just going to increase.

Afterwards, we need to be able to argue for our choice of databases and why we have chosen to assign them the functionality that we have.

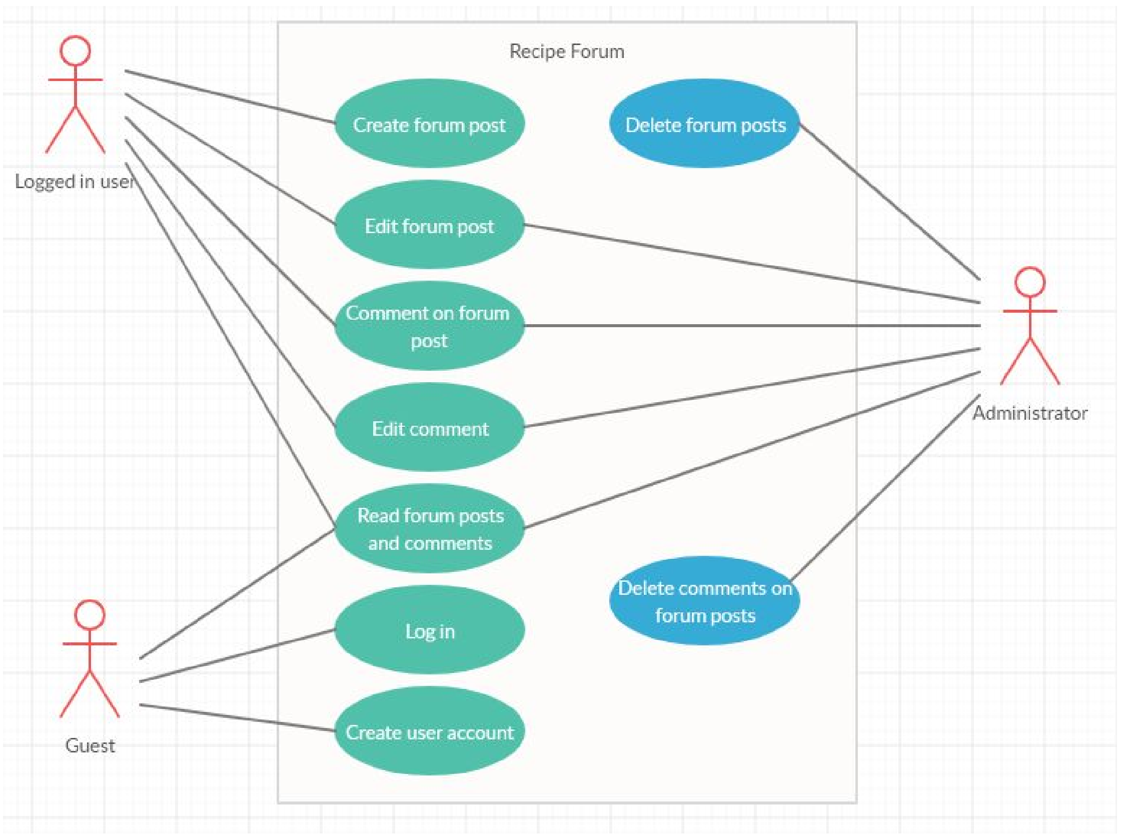
As a to-do, we are going to develop an application as a proof of concept, on top of that, we will be implementing the databases of our choice into said application. By doing so, we are going to be able to test the databases towards the application.

## Requirements

As any other project, we need to define some functional and nonfunctional requirements.

- Functional requirements:
    - Users can make posts, comments, edit either of them, delete their own posts and read other users posts and comments.
    - Guests on the forum can view posts and read comments.
    - Admins have all the same functionalities as the users, except they can delete whoevers posts and comments they deem to be inappropriate or out of context.
- Non-functional requirements:
    - Quick response times on each page load.

The picture below is how we visioned the forum's use cases.



# Boundary

This chapter will shortly describe the thing there will not be in this rapport.

The first thing we have chosen not to make is a frontend, that means the test of functionality will be tested using the 3rd party software “Postman”.

The main reason we have chosen not to make a frontend is the time needed to make one, and the fact that it is not really the main focus of the curriculum.

The different servers and the network used will neither be explained in this rapport, as the main focus is the databases. To give an example, we will use Ubuntu server 20.04 as the server to host the databases, there will not be any guides to install or how to use this server.

# Realization

## Methods (tools and software)

This chapter will go into detail about what tools and software we have used to make both the application and the databases. It will divide into 3 categories: Databases, Backend server and the test of the frontend

## Databases

This chapter will shortly describe the databases we have chosen to work with, and the server software the database is installed on.

Firstly we have chosen to install the different databases on the same server software “Ubuntu Server 20.04”.
Ubuntu is a free to use linux distribution, that are among the most used.

The servers is virtualized using “Hyper-V”.
“Hyper-V” is a “free” to use virtulations software made by Microsoft, the reason we put free in brackets, is that you do need to buy a windows client or server to use the software, but it comes as a free feature then.

The first database is “PostgreSQL” version 12.3.
“PostgreSQL” is a free open-source relational database, it has over 30 years of active development, and are widely used, therefore there is a lot of guides and help on the Internet

The second database is “MongoDB” version 4.2.6.

“MongoDB” is a general purpose, document-based and with support for distributed databases.

The third database is “Redis” version 6.0.3.
“Redis” is a key value databases, that uses the memory instead of the physical hard drive. This makes it very fast to find and write data.

## Backend server

This chapter will shortly describe the software and libraries used to create the backend server.

The programming IDE used to write the code for the backend server is “IntelliJ Idea”. This is developed by JetBrains.
This is a easy to use IDE, which also can be very advanced, so can be used both by beginners and professionals.

The programming language the backend server is written in, is Java, developed by Sun.
Java is meant as a language that is supported on every device.

There is also used a few plugins:

- “Jersey” version 3.0.0-M1.
    - Jersey handles the Rest api’s in the backend server.
- “Postgres” version 42.2.12.
    - Postgres is a driver to connect to the postgres database.
- “Mongo-java-driver” version 3.12.4.
    - Mongo-java-driver is a driver to connect to the MongoDB.
- “Jedis” version 2.6.2.
    - Jedis is a driver to connect to the Redis database.

## Test software

This chapter will very shortly describe the software used to test the application

The first and most important is “Postman”. “Postman” is a software the can do many things, in regards to you developed api’s.

The second software used is “Google Chrome”. “Chrome” is an free Internet browser.

# Analysis

This chapter will be about the databases we have chosen, and why we have chosen these databases.

We have decided to use Redis, MongoDB and Postgres as databases for our project. As for their functionalities, we have each one of them doing a specific task, such as store certain data.

## MongoDB

MongoDB database is the one containing all the posts, comments, recipes and nutrients.

There are a number of reasons we have chosen to use MongoDB to store most of the data.

- It is highly scalable, so if the need arise, it can easily be scaled to match the needs.
- It is document based, so it is easy to change it to the need of the application.
- It is very easy to use from the java sides. Inserting a document is handled on 1 line of code.
- It is used for a lot of different applications, which means there is a lot of guides on the Internet

## PostgreSQL

PostgreSQL are being used to hold the users login information, such as username, password and/or email.

There are a few reason why we have chosen to use PostgreSQL to handle this.

- PostgreSQL is a relational database, where the data is in a column. This makes it very good in our application to handle the user and their data, as this can be done in a single column. This makes looking up a user really fast. And can with a simple query find, if a user and password is a match.
- PostgreSQL have over 30 year of active development, so it is very reliable and is still being developed to this day.
- PostgreSQL is also used by many, meaning there is a lot of guides and info on the Internet.

## Redis

Redis is being used for the cache of the website. This is done to ensure the application can handle higher amount of requests per second.

There are a few reasons why we have chosen to use Redis.

- It is very fast, accessing the database in memory vs hard drive is the main reason for using Redis
- It is very simple to implement and program up against.
- There is a lot of guides on the Internet, meaning it's easy to get started using.

## Other databases we did consider

This chapter will be about the other databases we did consider, but opted not to use.

### HBase

We looked at Hbase, and did consider using it for storing the ingredients and nutrients, for each recipe. Where Hbase column based would work really well. You could have a column for ingredients and a second column for the nutrients.
There where a few reason why we did chose not to use Hbase.

- Hbase need to be large, if it’s not large, you might wanna look into another database
- Hbase is not very widely used, it is used by a lot of big companies, but very little information is out on the Internet. And a lot the that information is deprecated.
- We tried to find a Hbase driver for Java, and make a connection, af 3 hours, we chose not to continue down this path. (Not meaning we couldn't have done it, if needed be, just would take too long to learn it)

### Neo4j

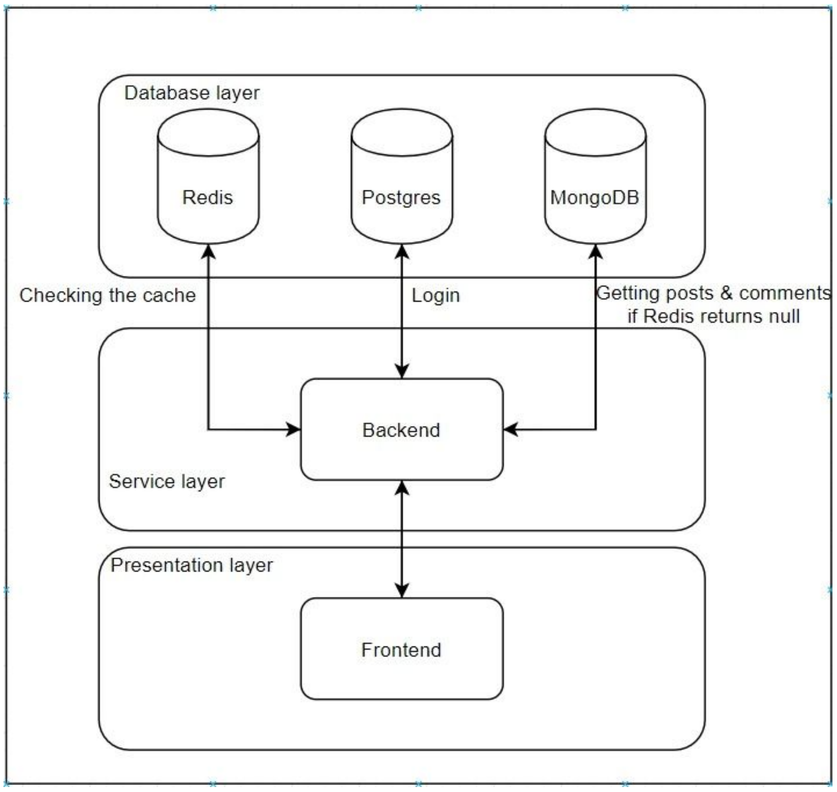
We did consider using Neo4j, but quickly opted not to use it. Neo4j’s graph database didn't really make much sense to use in this application. But might be considered in a future version, to find similar recipes.

# Architecture

This chapter will be covering the architecture used in the project.

The architecture for our application is a client server architecture, this architecture takes the concept that you have a client, that does not hold any data, and does not function on its own. It needs a server to connect to, that can supply it with data.

In our case, the backend is the server, and the frontend is the clients



We have also used a concept called Polyglot persistence[1].
The basics of the concept resolves around there being no database available to solve or tackle every single problem alone. The concept is therefore taking advantage of different aspects that a database is best at, to achieve the end goal.
The problem will be divided into smaller bits, and handed out to the databases best suited to solve that specific problem.

Another architecture used in our project is Representational State Transfer (REST), Application Program Interface (API). This architecture is providing a set of rules betweens systems on the web, making it easier for them to communicate with each other.

[1] https://www.objectrocket.com/blog/uncategorized/what-is-polyglot-persistence/

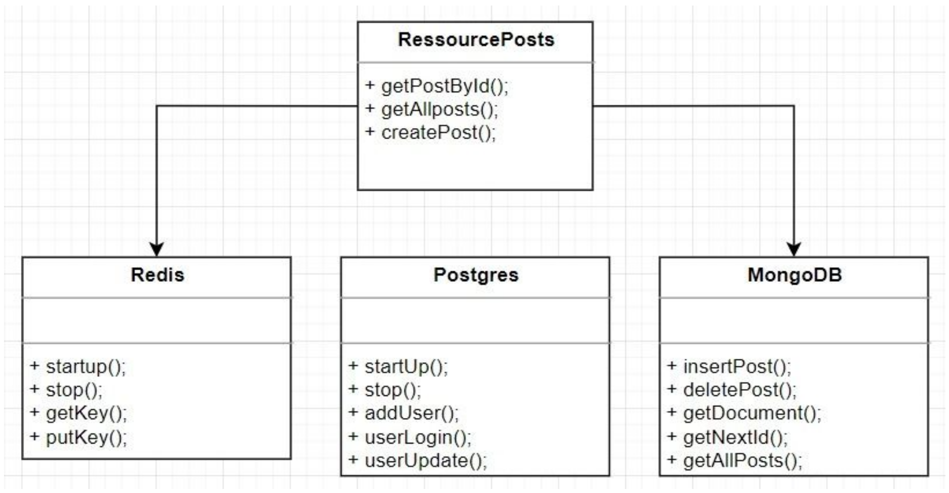
Restful system consists of a client who requests for the ressources, and a server who has the ressources the client will request. They are often characterized by their stateless and separate concerns for the client and the server.

# Design

This chapter is gonna contain in depth information about the application, what classes it is made up of, and how they work together. It also will explain the design of the documents used in MongoDB and PostgreSQL.

## Classes

The application is build up with classes in two different categories.



There are  3 Classes that are used only to communicate with the databases:

- MongoDB
- PostgreSQL
- Redis

These 3 classes each has different methods in them

PostgreSQL has 5 different methods.

- startUp()
    - This method is used at start, to start up the connection to the database
- stop()
    - This method is called before the application shuts down, and closes the connection to the database.
- addUser()
    - This method adds a new user to the postgreSQL database.

- userUpdate()
  - This method is used to update a user's email and password
- userLogin()
  - This method is used to validate a user login with a given username and password

Redis has 4 different methods.

- startUp()
  - This method starts ud the connection to the Redis database.
- stop()
  - This method stops the connection to the Redis database.
- putKey()
  - This method put a new key and a value into the Redis database. And set the expiration timer to 600 seconds.
- getKey()
  - This method retrieves a value from the Redis database. And at the same time set the expiration timer back to 600 seconds.

MongoDB has 5 different methods

- insertPost()
  - This method is used to create a new post in the MongoDB .
- deletePost()
  - This method is used to delete a post from the MongoDB.
- getDocument()
  - This method is used to get a post from the MongoDB
- getNextId()
  - This method is used to get the next available number, used to create a post in MongoDB
- getAllPosts()
  - This method returns all the documents in the collection posts, from the MongoDB.

There is currently 1 resource class that is used as the rest api, that uses the 3 different database classes.

ResourcePosts has 3 different methods

- getallposts()
  - This method returns all the posts in the MongoDB's collection 'posts'. It returns them as an url that if clicked on or followed will lead to that posts.

- getPostById()
    - This method returns a post based on its id. The method first queries the Redis database if it has that id(as a key). If Redis does not contain the key, the method then queries the MongoDB. After it gets the result from MongoDB, it put the post into the Redis database, and it will be there for the next 10 minutes.

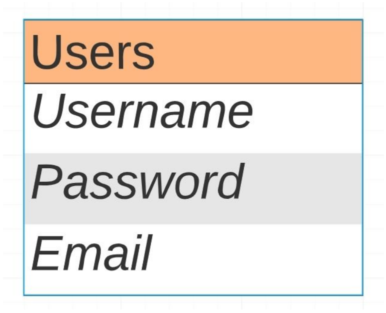
- createPost()
    - This method inserts a new document into the MongoDB. This method also gets the next "_id" value to be used by the post, and generates a timestamp for that post

## Database structure

The different databases has different way of holding their data. This chapter will explain how each database holds its data.

### PostgreSQL

The PostgreSQL database Is built simple with only 1 table called users, that contains 'username', 'password' and ' email.



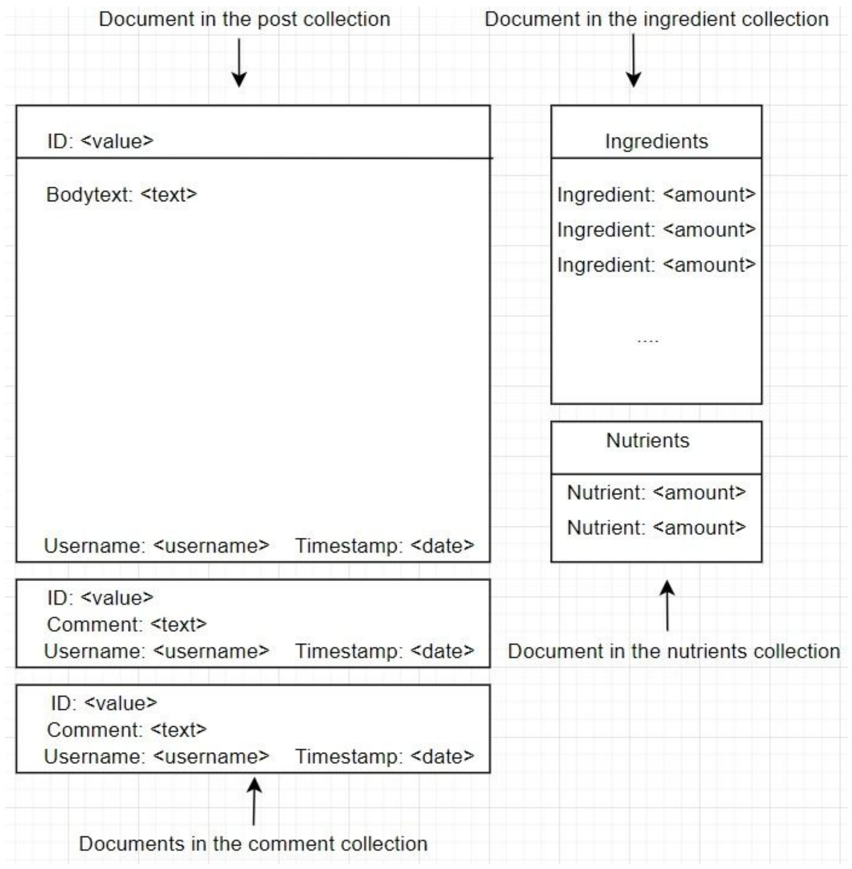
### Redis

The Redis database is used to cache the responses from the MongoDB as JSON string with the key of the posts "_id".

### MongoDB

This database is built up with 5 different collections.

- posts
- comments
- ingredients
- nutrients
- counter



# Test

This chapter will explain more about the different test made, to ensure the functional requirements.

We have used the classes 'main'  to conduct simple testing of the application's database classes, that the different classes behaved like they were supposed to.
We then used the 3rd party software 'Postman' to tests that the rest api's functions were behaving like they were meant to.

## Conclusion

We have successfully made a proof of concept, showing the capabilities of having the concept polyglot persistence in an application.
We didn't use 4 databases, but we did consider using more than the 3 we did use. But found that it would not have made sense forcing more into the application.
We did get a running environment where the different databases is hosted on different servers, and not all running on a single machine.

## Future work

In a future version, we could add Neo4j to make relations between each recipe, for instance based on how spicy the dishes are.

## References

https://www.postgresql.org
https://ubuntu.com
https://www.mongodb.com
https://redis.io
https://www.postman.com

# Appendixes

## Appendix 1 - Installation guide to the databases

- Install Redis
  - sudo apt install net-tools
  - sudo apt install make
  - sudo apt install gcc
  - wget http://download.redis.io/releases/redis-6.0.3.tar.gz
  - tar xvzf redis-6.0.3.tar.gz
  - cd redis-6.0.3.tar.gz
  - make
  - sudo cp redis-server /usr/local/bin
  - sudo cp redis-cli /usr/local/bin
  - sudo mkdir /etc/redis
  - sudo mkdir /var/redis
  - sudo cp utils/redis_init_script /etc/init.d/redis
  - sudo cp redis.conf /etc/redis/6379.conf
  - sudo nano /etc/redis/6379.conf
    - “daemonize yes”
    - pidfile /var/run/redis_6379.pid
    - “logfile "/var/log/redis_6379.log"
    - dir /var/redis/6379
    - requirepass password
    - #bind 127.0.0.1 //comment bind, so it listens on all ports
  - sudo update-rc.d redis defaults
  - sudo /etc/init.d/redis start

- Install MongoDB
  - wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc | sudo apt-key add -
  - echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
  - sudo apt-get update
  - sudo apt-get install -y mongodb-org
  - sudo systemctl start mongod

```
db.createUser(
  {
    user: "backend",
    pwd: "password",
    roles: [
       { role: "readWrite", db: "recipepage" }
    ]
  }
)
use recipepage
db.counter.insert({ "_id" : "itemId", "seqValue" : NumberInt(0) })
```

- postgreSQL
  - sudo apt install net-tools
  - sudo apt install postgresql-12
  - sudo nano /etc/postgresql/12/main/pg_hba.conf
    - host    all    all    192.168.1.0/24    md5
  - sudo nano /etc/postgresql/12/main/postgresql.conf
    - listen_addresses = '*'

- commands in psql
  - CREATE USER backend WITH PASSWORD ‘password’;
  - GRANT ALL PRIVILEGES ON DATABASE recipepage TO backend;
  - GRANT USAGE ON SCHEMA public TO backend;
  - GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO backend;
  - GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO backend;