**System Design of Customer Management System**

# Functional Requirements

We will design a basic customer relationship management software that keeps track of customer data, transactional data, and customer/employee interactions.

So what are some of the general goals for our basic customer relationship management software?

* Keep track of customer contact details, such as name, email, and phone
* Keep track of customer transactions with business
* Keep track of customer communications with business, whether it be for support, notifications, or inquiries

We can thus define some functional requirements as such:

## **Business Side**:

* Register an employee into the company database
* Send notifications regarding support status, etc. to customers
* Open a communication channel with a customer or another employee

## **Customer Side**:

* Register a customer into the company database
* Begin a conversation with an employee, through phone/email/chat
* Rate a conversation and provide feedback
* Purchase goods/services

# Non-Functional Requirements

* Scalability

Our software must be able to adapt to growing demands (i.e. our customer base increases significantly), meaning the software must be maintainable and adaptable. In addition, any new incoming developers must be able to look into the code when integrating features or solving bugs.

* Performance

Any queries that take place from employees or customers must be served in the most efficient way. Time is money, after all!

* Usability

Usability will be key, as any customer needing assistance with various inquiries or processes will need to be able to navigate and find help easily.

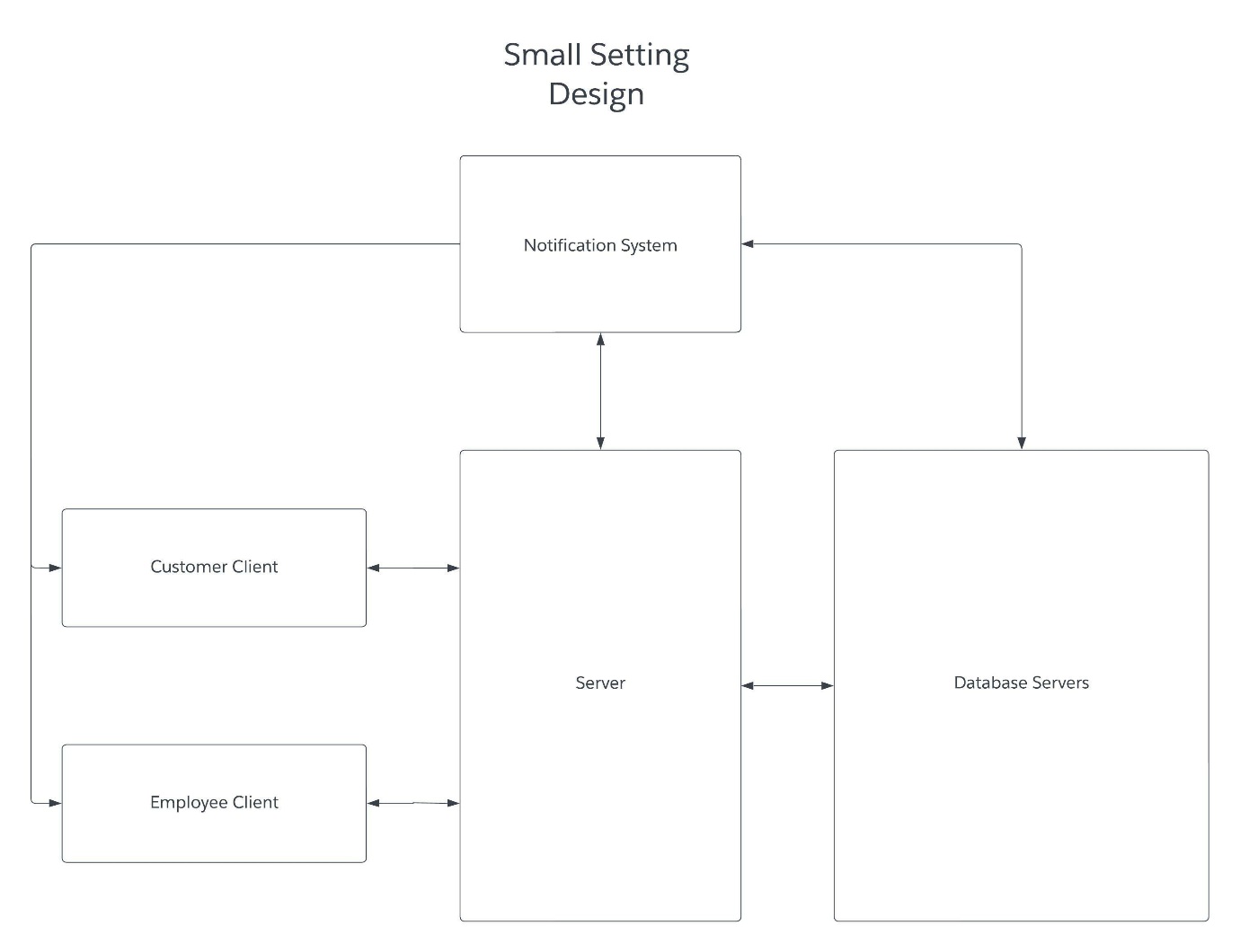
# Software Requirements

* Server cluster(s), operating in either Windows or Linux
* Relational database management system, preferably PostgreSQL as explained later
* Front-end frameworks, such as React or Angular
* Back-end server frameworks, such as Spring Boot
* Hosting technologies (i.e. Heroku, AWS, DigitalOcean)
* Load-balancing tools, such as AWS

# High Level Design

What, then, could our system look like for different kinds of environments?

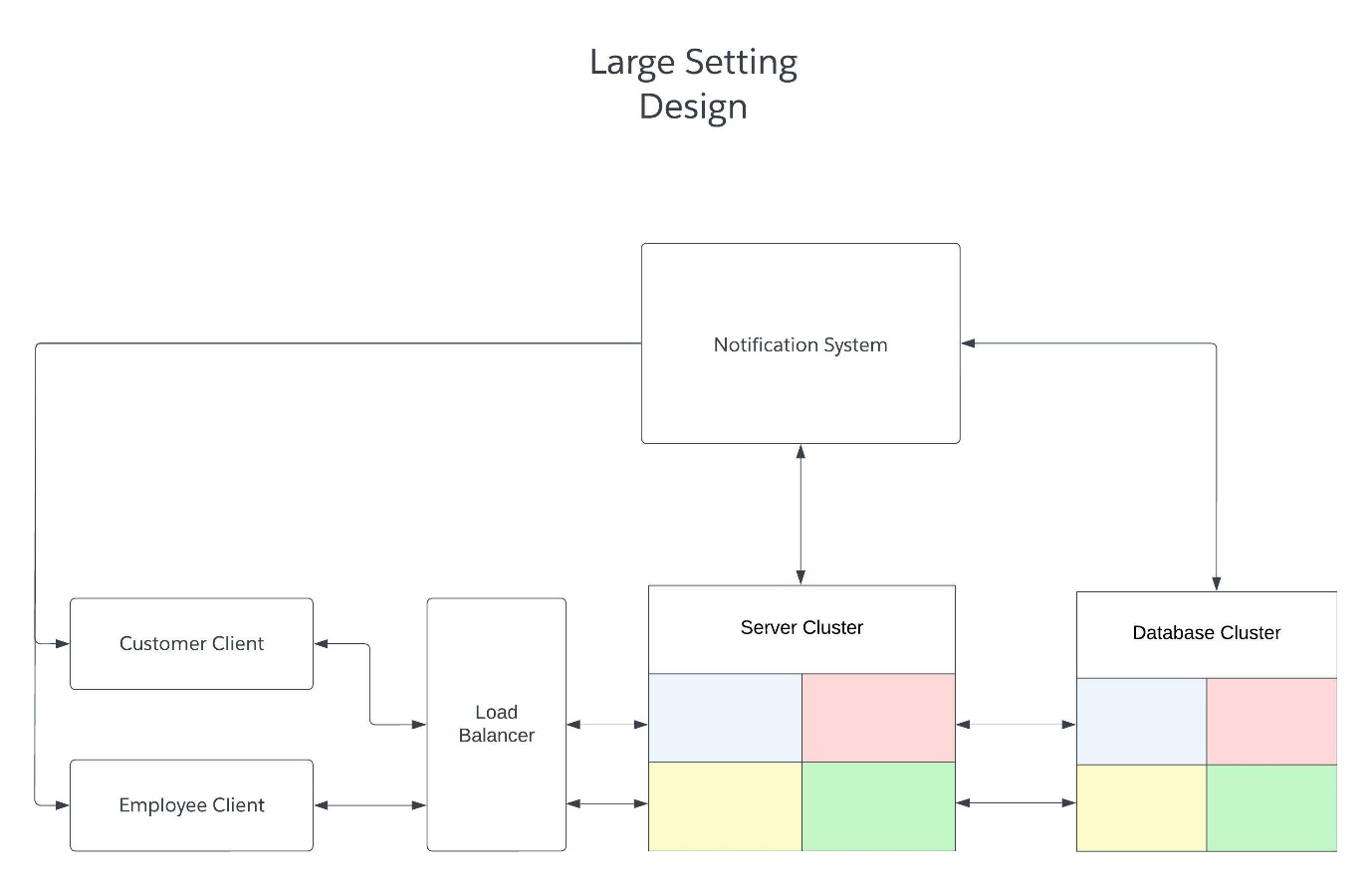
## **Small Business**

  
For a small business (i.e. a few thousand users), the traffic is relatively low, with at most a few thousand concurrent users. However, given that the business may also go through spikes over the year (i.e. holiday shopping, etc), it is still recommended that a business invest in a group of servers, preferably with 16 GB RAM and good processors each.

To help deal with many requests at once, a request queue or list needs to be implemented to choose a request to serve. Since our customer base is only about a few thousand, the request queue should be enough to compile the incoming requests.

Customers can log in to the application and request to open communication channels with employees. In turn, employees can log in to the application and open such communication requests, providing assistance when possible.

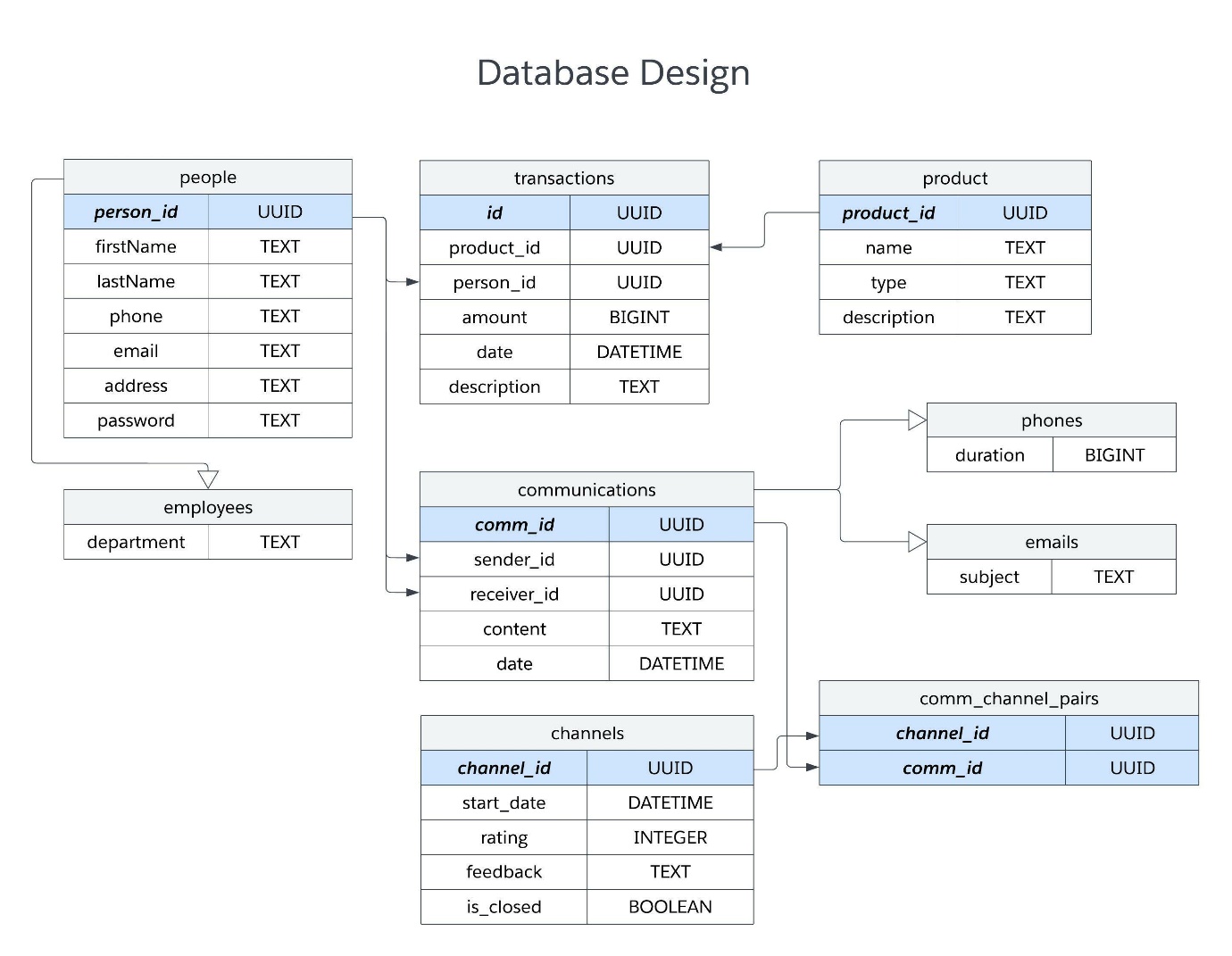
## **Large Business**

  
With larger businesses (i.e. tens or hundreds of thousands of users or more), it will become necessary to upgrade the hardware, as well as distribute hardware resources. In our case, we may need to start by investing in multiple server clusters. We may need to begin sharding our databases into database clusters -- that is, distribute parts of our database to different clusters to ease the load off of data input and output operations. Perhaps, for instance, we can store transactions in different clusters by month of year or by region. Many companies such as Instagram make use of PostgreSQL to help shard their databases (see my previous article on PostgreSQL); doing so will be natural with our preferred choice of PostgreSQL as our database.

However, dealing with more requests will become significantly harder, as the server must now deal with tons or hundreds of thousands of users, or more (and thus, dealing with millions of requests becomes the norm). In such a case, it will be necessary to begin relying on multiple servers and/or server clusters. To balance incoming requests, a load balancer will be necessary to reroute traffic to different servers, perhaps by region or by server capacity. In any case, this load balancer will ensure that a single server is less likely to fail because of a sudden spike in (or a consistently high number of) requests.

# Database Design

We can design our database(s) like so:



Generally speaking, the tables contain the same properties as their respective Java classes (with rows highlighted in blue as primary keys). For instance, the people table has a primary key **person\_id** with the same properties as the Person class in Java (with the exception of password, which is necessary for any user who needs to log into a system).

Some tables, such as employees, phones, and emails, have the same properties as other tables, such as people and communications. Such tables would necessitate table inheritance, as without table inheritance, the database would become far more complicated, having to keep track of two nearly identical tables (i.e. employees and people) with the only difference being a few properties. The relational solution to this problem is PostgreSQL, which has table inheritance as a feature.

# Conclusion

In conclusion, we've made a basic design for a customer relationship management software. While certain features may vary depending on the client, the general necessities of CRM software should be implemented in this guide.