Technology Research Document

Name: Tony Jiang

Semester: 6

Class: RB04

Contents

Introduction 3
Technology for backend (individual) 3
Database selection (individual) 3
CAP theorem 3
Polyglot persistence 4
Text file store (group) 5
Cloud technology (individual) 6

Introduction

This document contains all the research I've done on the technology and methods I use, along with the reasons for my choices. This way, I can showcase to the teachers that I have explored the options and justified my decisions. This research can be for the individual project or the group project.

Technology for backend (individual)

This semester, I have the opportunity to explore a new programming language. A friend recommended that I try Golang, so I did some research. Initially, I was planning to use Java because of its vast library options, built-in security features, high scalability, and strong support for microservices.

However, Golang (Go), which was designed by Google, also offers several appealing features. It has a simple and easy-to-read design, fast compilation, strong support for microservices, and is excellent for cloud-native development. Additionally, it can easily scale horizontally and is highly favored for DevOps and infrastructure tasks. The only problem is that I need to learn how to code in Golang.

The obvious choice is Golang because it adheres to all the learning outcome of this semester.

Database selection (individual)

Choosing the right database for a project can be challenging. To make this decision, I use the CAP theorem and it is reference Canvas.

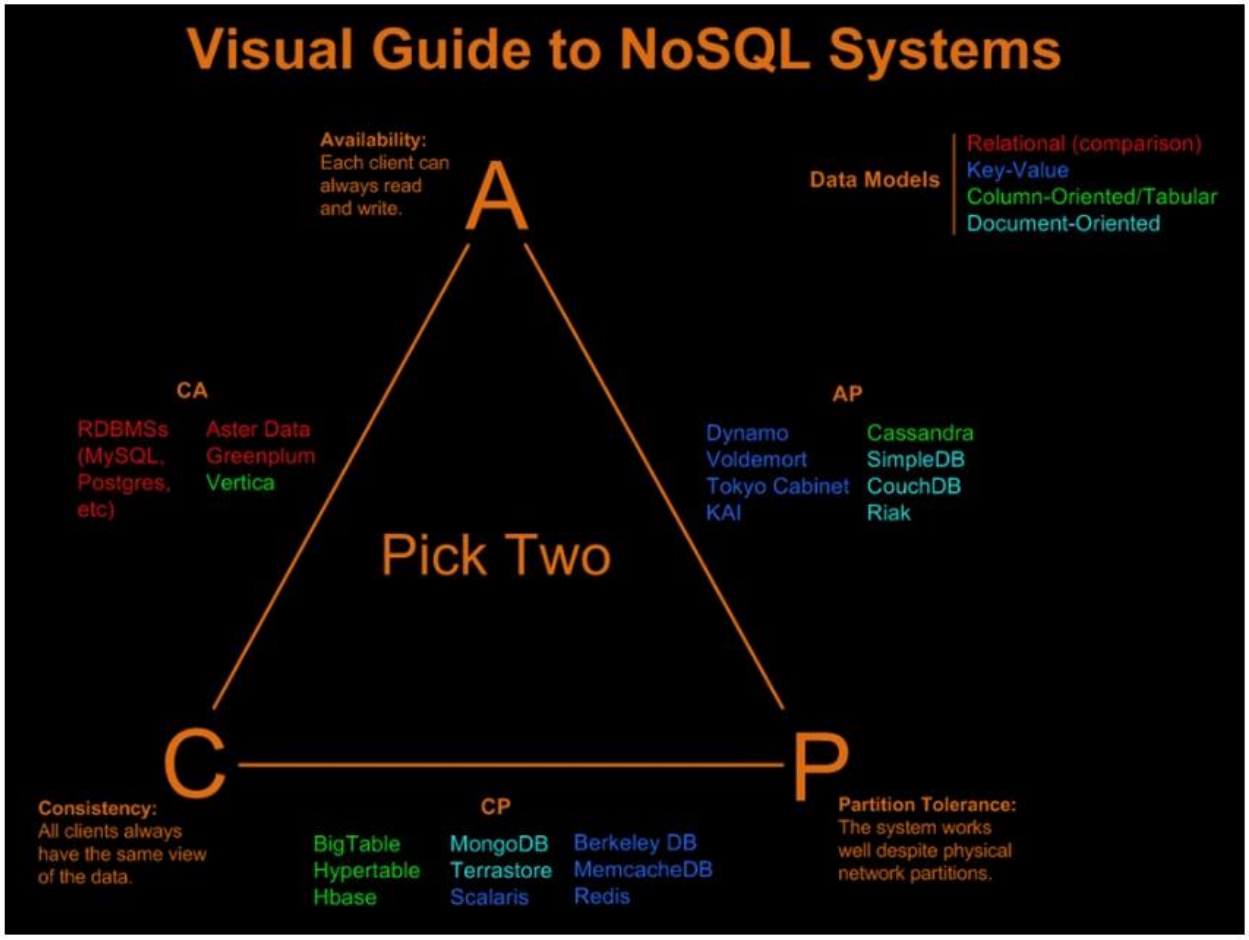
CAP theorem

The CAP theorem is a fundamental concept in distributed database systems. It states that it is impossible for a distributed data store to simultaneously guarantee all three of the following properties:

- Consistency (C): Every read receives the most recent, correct, and up-to-date information, regardless of which server (or node) you ask.
- Availability (A): Every request (read or write) receives a response, even if it doesn't guarantee that the response contains the most recent write.
- Partition Tolerance (P): The system remains operational despite network partitions or communication breakdowns between nodes.

According to the CAP theorem, you can only guarantee two out of these three:

- CP (Consistency + Partition Tolerance): The system remains consistent and tolerates network partitions, but this comes at the cost of availability—some requests may fail.
- AP (Availability + Partition Tolerance): The system prioritizes responsiveness and handles network issues, but consistency may be sacrificed, meaning you might not always receive the latest data.
- CA (Consistency + Availability): The system ensures both accurate, up-to-date data and immediate responses, but it cannot tolerate network failures or partitions.



Based on the CAP theorem, my project should prioritize AP (Availability + Partition Tolerance) to ensure quick response times and resilience to network issues. My users prefer faster responses over strict data consistency since they are playing a game, and they value immediate feedback on their answers more than having perfectly consistent data. Therefore, my system will need an AP-focused database. A reference to AP databases is provided in the image above.

Polyglot persistence

In microservices architecture, it's common to use Polyglot Persistence, where each microservice can use the database best suited to its specific needs. You don't need to rely on a single type of database for the entire system. Different microservices can use different databases based on their requirements.

Text file store (group)

I was tasked with creating a text file storage solution for the botlist service for the group. Since I had no prior knowledge or experience in this area, I conducted some research on how to implement it. The language used to create the botlist service is Java, and the text file can be stored in JSON format.

I explored possible solutions for storing data in a text file using Java and identified two dependencies that could meet my requirements:

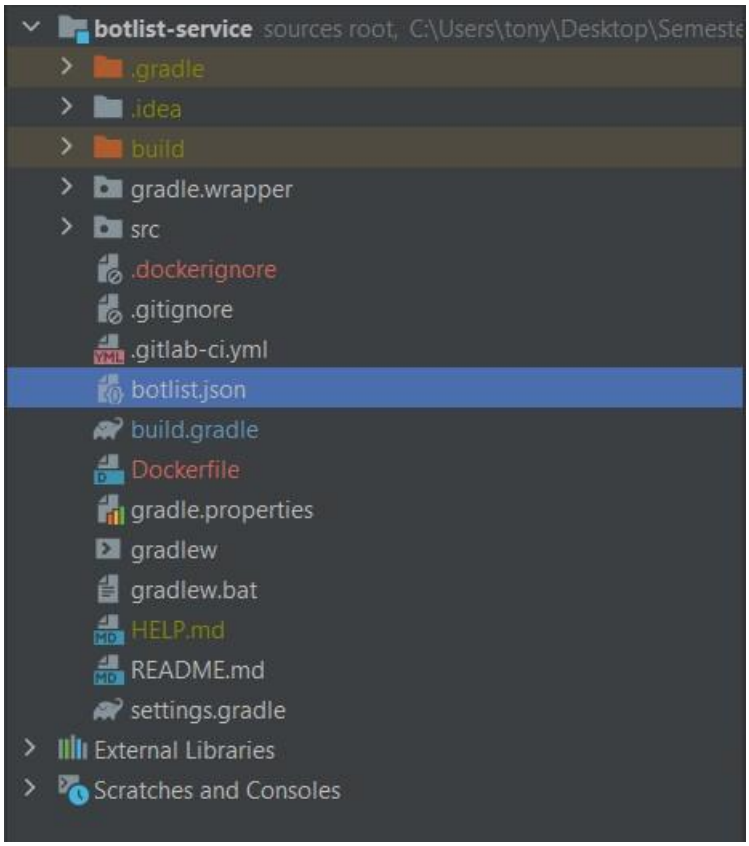
the Gson library and the Jackson library.

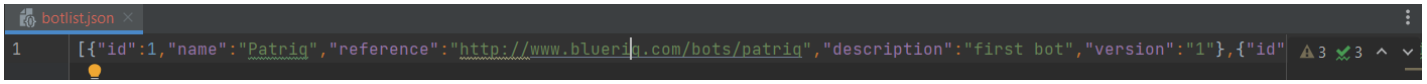
I chose the Jackson library due to its advanced features, better performance, and support for polymorphism. This choice also allows for future scalability, ensuring high performance and accommodating larger datasets if the company decides to expand.

While the Gson library is simpler and more beginner-friendly, I opted for Jackson to better suit the project's needs. All you need to do is add the Jackson dependency in build.gradle, and you can start coding.

```
dependencies {
    //implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    //testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    // https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind
    implementation 'com.fasterxml.jackson.core:jackson-databind:2.18.1'
```

The text file is stored in the root folder of the botlist service. If the file does not exist, the system will automatically create it when a botlist is added. However, the file does not track IDs, as it lacks an auto-increment mechanism like a database, so an auto-increment feature will need to be implemented manually.





Cloud technology (individual)

For the cloud technology that I'm going to use is either Kubernetes or cloud function. I have a bit more experience with Kubernetes than cloud function. Cloud function is very new to me, I know that with cloud function instead of deploying your microservice into the cloud you can deploy a function to the cloud.