Faculty of
Information Technology
King Mongkut's Institute of Technology Ladkrabang

# Chapter 07

# Caching Data for Microservice


บรรยายโดย ผศ.ดร.ธราวิเชษฐ์ ธิติจรูญโรจน์ และอาจารย์สัญชัย น้อยจันทร์

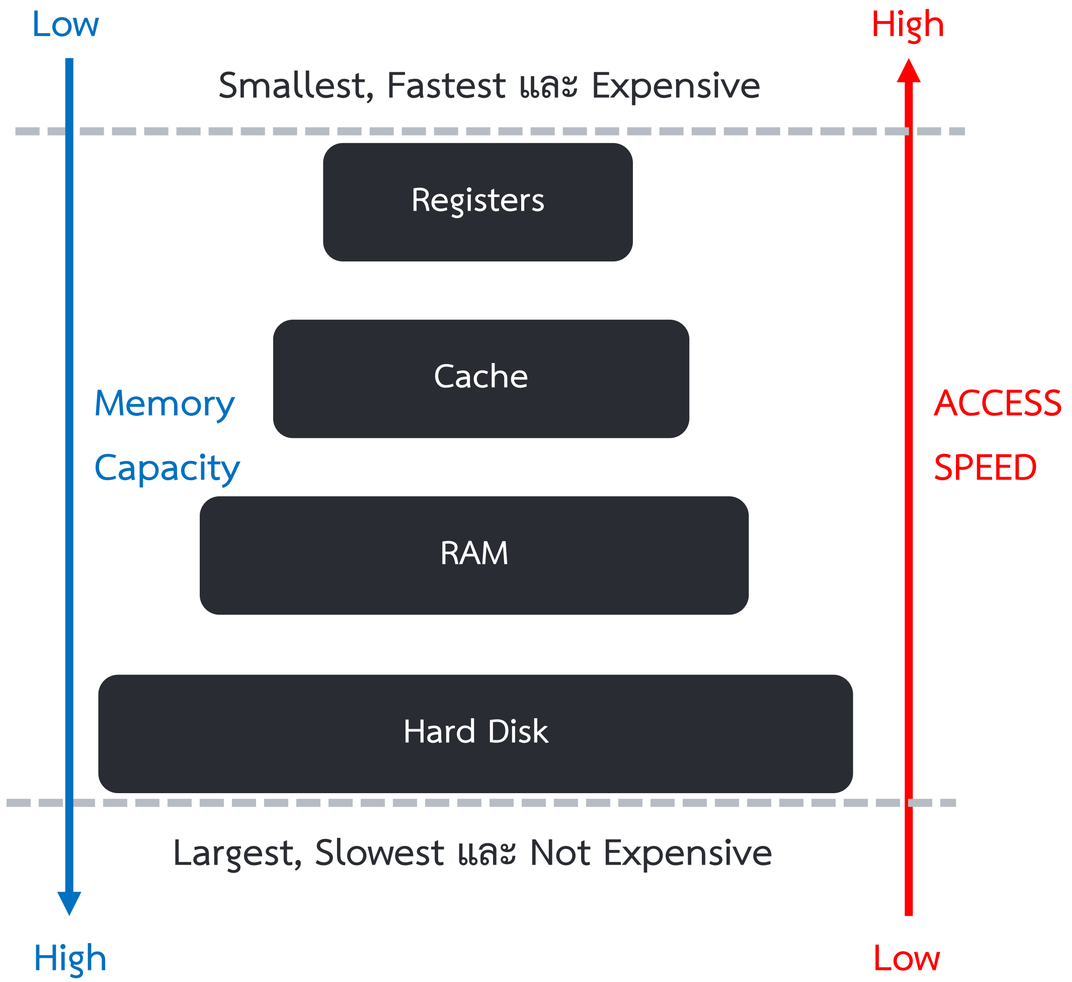
คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง


# Outline

- แนวทางการออกแบบระบบด้วย Repository Pattern
- แนวคิดของฐานข้อมูลแบบ NoSQL และการใช้งานฐานข้อมูล MongoDB
- การโปรแกรม Spring Boot สำหรับเชื่อมต่อฐานข้อมูล MongoDB แบบ Repository Pattern
- การแคชข้อมูล (Caching) ด้วย Redis

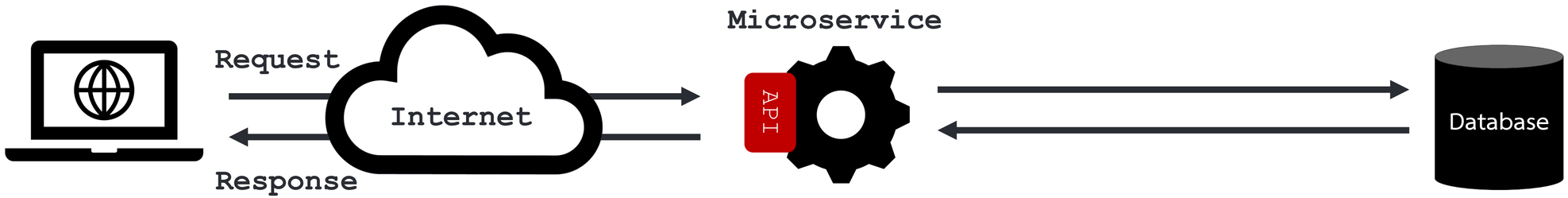
# Buffer และ Caching คืออะไร




Cache คือ memory (อาทิเช่น Ram) หรือ data storage (อาทิเช่น HDD) ที่ใช้เก็บข้อมูลที่มีการเรียกใช้งานบ่อย เพื่อเพิ่มประสิทธิภาพด้านความเร็วในการให้บริการ

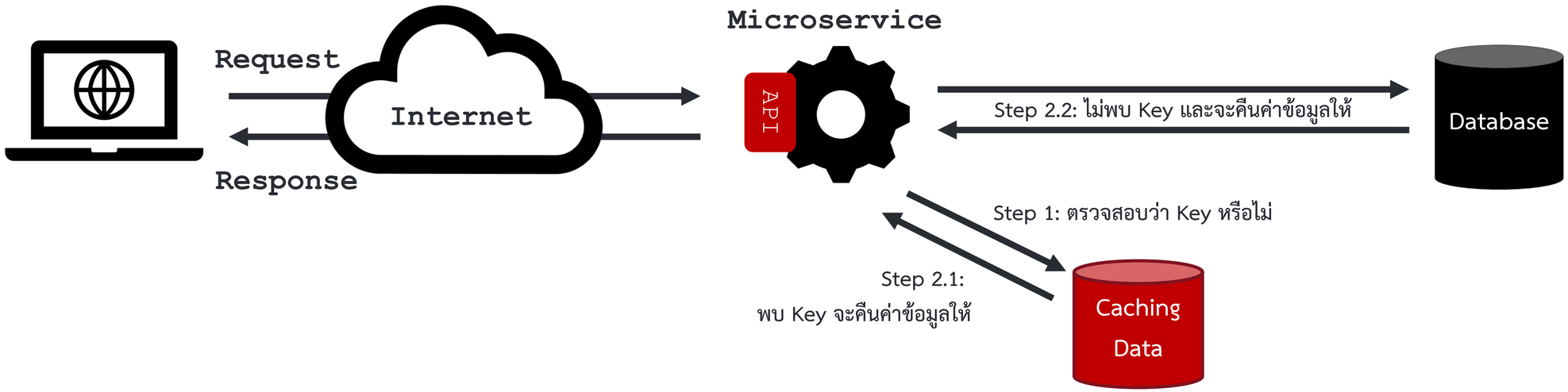
Buffer คือ memory หรือ data storage ที่นำมาช่วยชดเชยความต่างกันของเวลาและความเร็วในการแลกเปลี่ยนข้อมูลระหว่าง Process กับ Device

# Caching Data คืออะไร



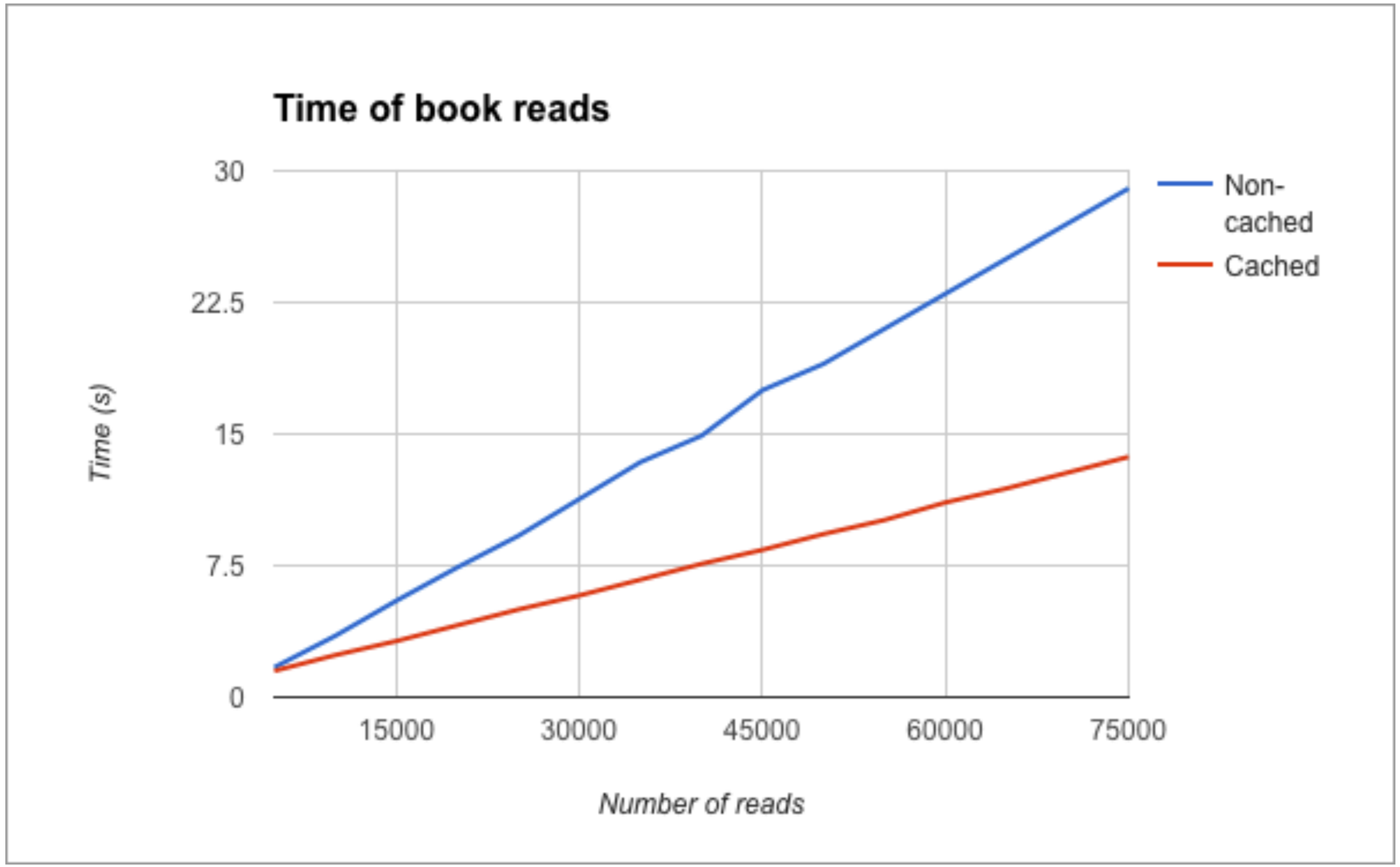

การร้องขอใช้บริการจากโปรแกรม (หรือ Service) อื่น ๆ ในรูปแบบ API เพื่อแลกเปลี่ยนข้อมูลระหว่างโปรแกรม ถือว่าเป็นเรื่องปกติสำหรับการพัฒนาโปรแกรมแบบ Microservice แต่การร้องขอข้อมูลตัวเดิมซ้ำ ๆ และบ่อยครั้ง อาจส่งผลเสียในเรื่องของเวลาในการอได้ ตัวอย่างเช่น ถ้านักศึกษาร้องขอจำนวนผู้ติดเชื้อโควิดในวันนี้ แน่นอนว่าทุกการร้องขอภายในวันนี้จะได้ข้อมูลที่เหมือนกันไม่ว่าการร้องขอจะมาจากใคร นอกจากนี้ ข้อมูลมีการจัดเก็บในฐานข้อมูลซึ่งจะใช้เวลาในการประมวลผลและตอบกลับโดยเฉลี่ยอยู่ที่ 10,000 ถึง 15,000 มิลลิวินาที

# Caching Data คืออะไร



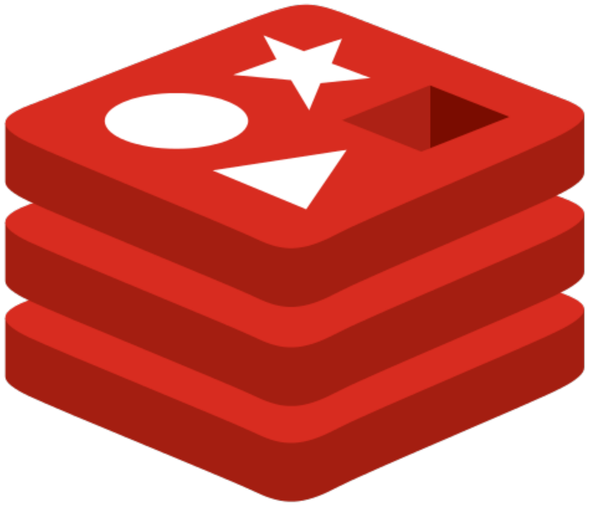

ถ้าต้องการลด Load Time ลงสำหรับข้อมูลที่มีการเรียกใช้งานบ่อย สามารถทำได้ด้วยการเทคนิคที่เรียกว่า Caching Data ซึ่ง Caching Data เป็นกระบวนการจัดเก็บสำเนาข้อมูล (Temporary Storage หรือ Cache) ที่ช่วยให้เข้าถึงข้อมูลได้เร็วขึ้น ส่งผลทำให้ใช้เวลาลดลงได้โดยเฉลี่ยอยู่ที่ 7,000 ถึง 10,000 มิลลิวินาที

# Caching Data คืออะไร




https://www.cnblogs.com/princessd8251/articles/6547964.html

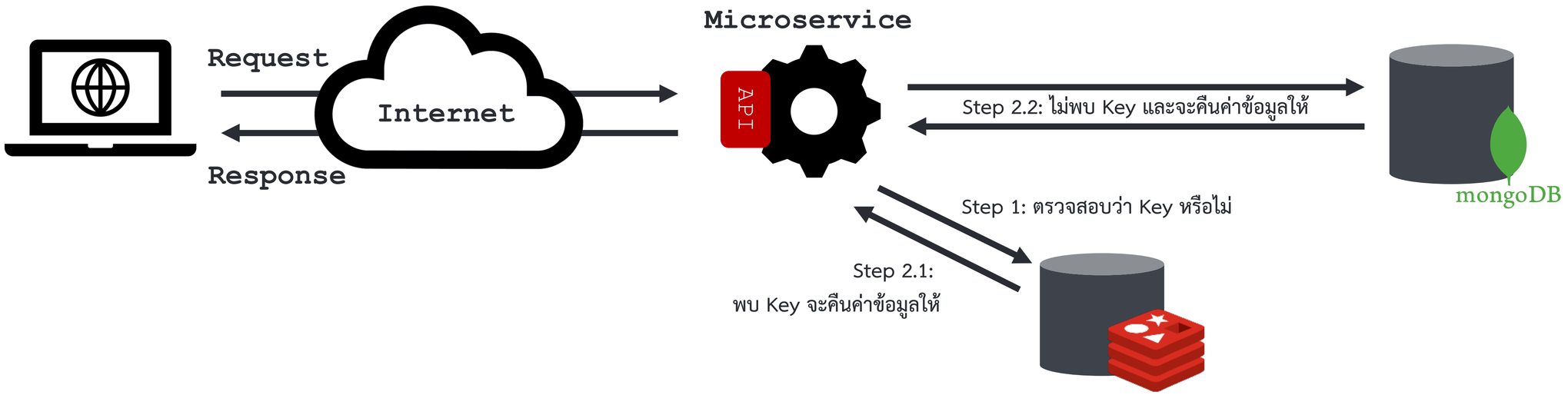
# Redis คืออะไร

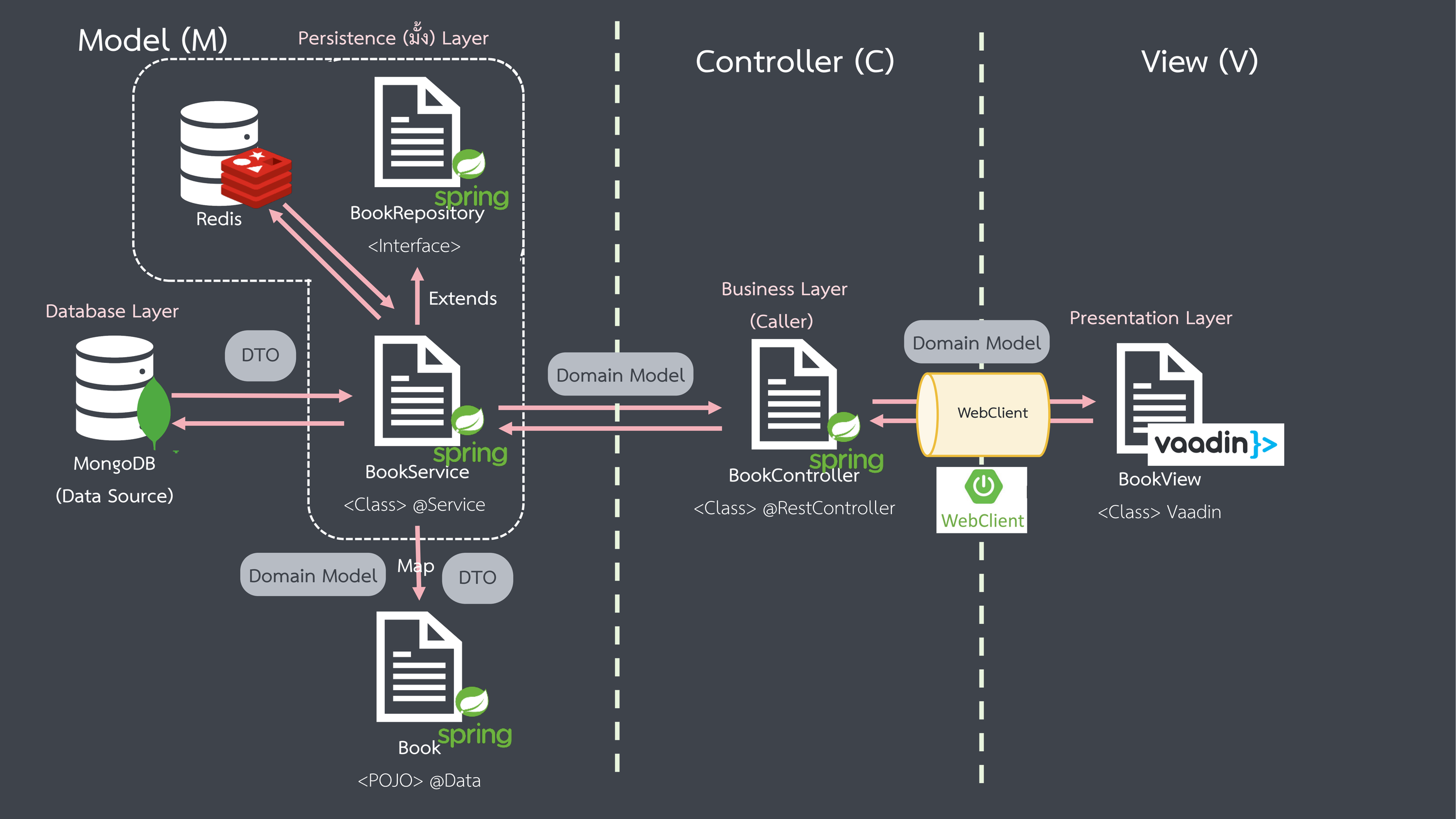


redis

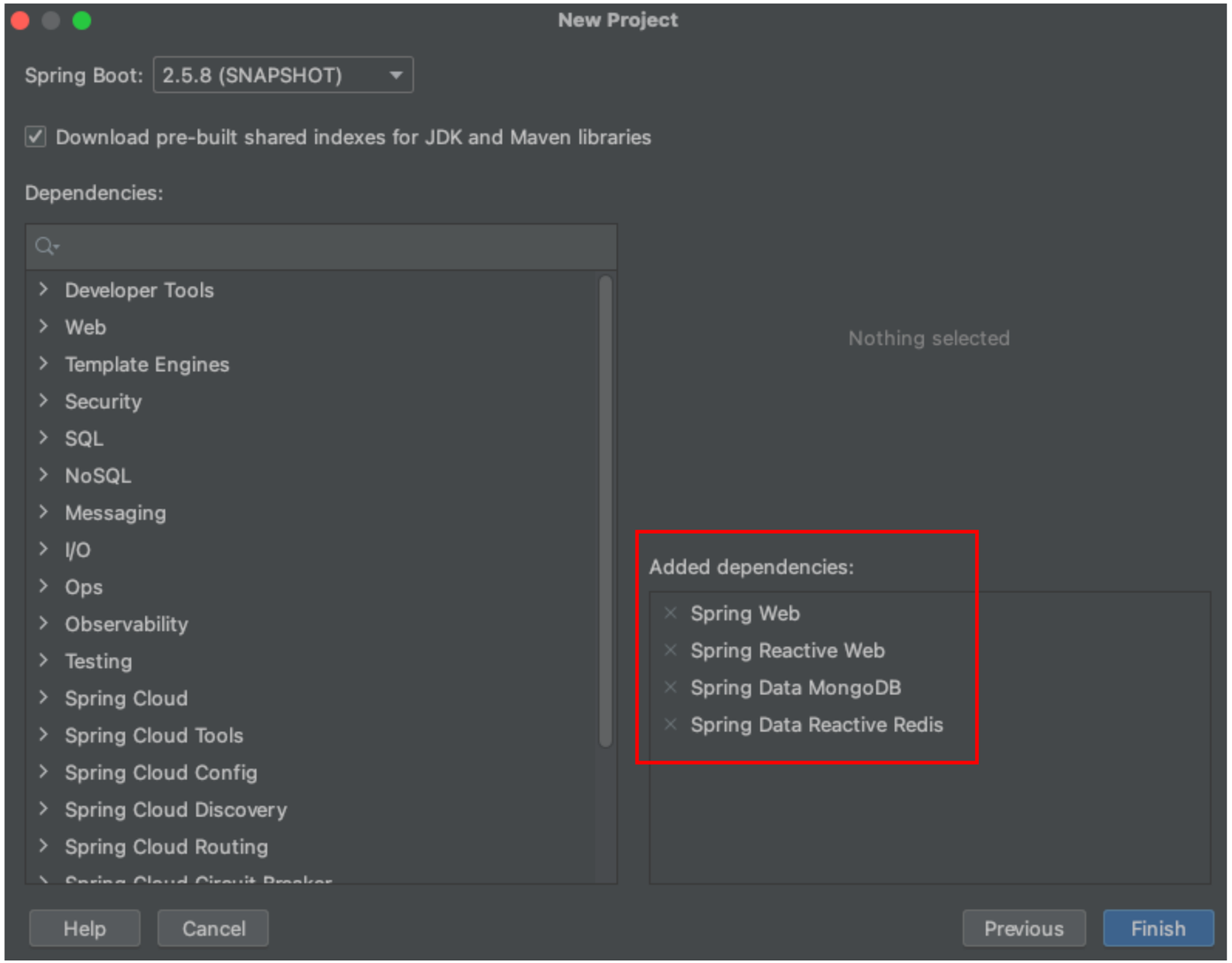
เป็น Open Source ที่ใช้จัดเก็บข้อมูลและโครงสร้างข้อมูลแบบ Memory (มีความคล้ายคลึงกับฐานข้อมูลประเภท NoSQL) ซึ่งครอบคลุมการทำงานในส่วนของ (1) การเก็บข้อมูล (2) แคช และ (3) Message Broker ซึ่ง Redis รองรับโครงสร้างข้อมูลดังต่อไปนี้ String, Hashes, Lists, Sets, Sorted sets, Range Queries, Bitmaps, Hyperloglogs, Geospatial Indexes, and Streams

# โครงสร้างการทำงานระหว่าง Redis, MongoDB และ Spring Boot




Model (M)
Persistence (มั้ง) Layer
Controller (C)
View (V)
Redis
BookRepository
<Interface>
spring
Extends
Database Layer
DTO
MongoDB
(Data Source)
BookService
<Class> @Service
Domain Model
Map
DTO
Book
<POJO> @Data
Business Layer
(Caller)
Domain Model
BookController
<Class> @RestController
Domain Model
WebClient
WebClient
Presentation Layer
vaadin
BookView
<Class> Vaadin

# ขั้นที่ 1 การจัดการ Dependencies




# ขั้นที่ 2 กำหนดค่า application.properties

```
// กำหนดที่อยู่ของ Database
spring.data.mongodb.host=localhost

// กำหนด Port ที่ใช้เชื่อมต่อกับฐานข้อมูล
spring.data.mongodb.port=27017

// กำหนดชื่อฐานข้อมูลที่จะติดต่อไป
spring.data.mongodb.database=SOA

// กำหนดตัวที่ทำหน้าที่ Caching Data
spring.cache.type=redis

// กำหนดที่อยู่
spring.redis.host=localhost

// กำหนด Port ที่ใช้เชื่อมต่อ
spring.redis.port=6379
```


# ขั้นที่ 3 สร้าง POJO (หรือ Entity Class)

```
@Data // เพื่อบ่งบอกว่าคลาสนี้เป็น Plain Old Java Object (POJO)
@Document("Book") // เพื่อกำหนด collection name ที่คลาสนี้จะเป็น Data Model
public class Book implements Serializable {
    @Id // ใช้บ่งบอกว่าแอททริบิวใดเป็นคีย์ และสร้างให้รองรับ ObjectId ของ MongoDB
    private String _id;
    private String name;
    private String category;
    private int price;

    public Book() {}
    public Book(String _id, String name, String category, int price) {
        this._id = _id;
        this.name = name;
        this.category = category;
        this.price = price;
    }
}
```


# ขั้นที่ 4 สร้าง Repository แบบ Interface

```
@Repository
public interface BookRepository extends MongoRepository<Book, String> {

    @Query(value="{name:'?0'}")
    public Book findByName(String name);

    @Query(value="{name:'?0'}", fields = "{'_id':0,'price':1}")
    public Book findByName2(String name);

    @Query(value="{price: {$gt:?0}}", count = true)
    public Integer countMoreThanX(int num);

}

// เหมือนกับตอนยังไม่เชื่อมต่อ Redis ดังนั้น ไม่ต้องแก้ไขเพิ่มเติม
```


# ขั้นที่ 5 สร้าง Service (หรือ RepositoryImpl)

```
@Service
public class BookService {
    @Autowired
    private BookRepository repository;
    public BookService(BookRepository repository) {
        this.repository = repository;
    }
    @Cacheable(value="mybooks”)
    public List<Book> retrieveBooks() { return repository.findAll();   }
    @Cacheable(value="mybooks")
    public Book retrieveBookByName(String name) {
        return repository.findByName(name);
    }
    @Cacheable(value="mybooks")
    public Book retrieveBookByName2(String name) {
        return repository.findByName2(name);
    } // ต่อหน้าถัดไป
```


# ขั้นที่ 5 สร้าง Service (หรือ RepositoryImpl)

```
    public Book createBook(Book book) {
        return repository.save(book);
    }
    @CacheEvict(value="mybooks")
    public boolean deleteBook(Book book) {
        try { repository.delete(book);  return true;  }
        catch (Exception e){ return false;}
    }
    public int countBook() {
        return (int)repository.count();
    }
    public int countBookPriceMoreThanX(int x) {
        return repository.countMoreThanX(x);
    }
    @CachePut(value = "mybooks")
    public Book updateBook(Book book) {
        return repository.save(book);
    }
}
```


# การใช้งาน Caching แบบ Annotations

The simplest way to enable caching behavior for a method is to demarcate it with @Cacheable, and parameterize it with the name of the cache where the results would be stored:

```
@Cacheable("addresses")
public String getAddress(Customer customer) {...}
```

The getAddress() call will first check the cache addresses before actually invoking the method and then caching the result. While in most cases one cache is enough, the Spring framework also supports multiple caches to be passed as parameters:

```
@Cacheable({"addresses", "directory"})
public String getAddress(Customer customer) {...}
```

In this case, if any of the caches contain the required result, the result is returned and the method is not invoked.

อ้างอิง https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/cache.html


# การใช้งาน Caching แบบ Annotations

Put @Cacheable on the method which you want to cache.  @Cacheable annotations has 3 attributes

- Value : is the cache name and it is mandatory, in example it is “messagecache”
- Key: based on this data will be cached and it is optional
- Condition:  based on the condition data will be cached. In example if the id < 10 then only data will be cached otherwise won’t. it is optional

```
@Cacheable(value="messagecache", key="#id", condition="id < 10")
 public String getMessage(int id){ ...  }
```

Here getMessage() method is marked with @Cacheable, whenever getMessage() is called it will check the messagecache, if the data is already in messagecache it will return that otherwise it will executes the getMessage() and returns data.

อ้างอิง https://java2practice.com/2013/03/23/spring-cacheable-and-cacheevict-explained-in-simple-terms/


# การใช้งาน Caching แบบ Annotations

@CacheEvict annotation will be used to delete the data from existing cache. The problem is size. We don't want to populate the cache with values that we don't need often. Caches can grow quite large, quite fast, and we could be holding on to a lot of stale or unused data.

There are 5 attributes:

- Value,  Key, condition -> are similar to @Cacheable, apart from these 3 we have another 2 attributes
- allEntries : is a Boolean type and delete entire cache
- beforeInvocation: is Boolean type and will delete the cache before the method execution

Here whenever a saveEmployee() is called cache will be deleted.

```
@CacheEvict("employees")
public void saveEmployee(Employee e){...}
```

อ้างอิง https://java2practice.com/2013/03/23/spring-cacheable-and-cacheevict-explained-in-simple-terms/


# การใช้งาน Caching แบบ Annotations

While @CacheEvict reduces the overhead of looking up entries in a large cache by removing stale and unused entries, we want to avoid evicting too much data out of the cache. Instead, we selectively update the entries whenever we alter them. With the @CachePut annotation, we can update the content of the cache without interfering with the method execution. That is, the method will always be executed and the result cached:

```
@CachePut(value="addresses")
public String getAddress(Customer customer) {...}
```

The difference between @Cacheable and @CachePut is that @Cacheable will skip running the method, whereas @CachePut will actually run the method and then put its results in the cache.

อ้างอิง https://www.baeldung.com/spring-cache-tutorial/


# ขั้นที่ 6 สร้าง RestController

```
@RestController
public class BookController {
    @Autowired
    private BookService bookService ;

    // เหมือนกับตอนยังไม่เชื่อมต่อ Redis ดังนั้น ไม่ต้องแก้ไขเพิ่มเติม

}
```


# วิธีการติดตั้ง Redis

**วิธีการติดตั้งสำหรับ Windows**

- **วิธีการที่ 1 : ดาวโหลด .msi ของ redis มาติดตั้ง (ข้อเสีย จะได้รับรุ่นที่ค่อนข้างเก่า) ตามลิงค์ด้านล่าง**

  https://github.com/microsoftarchive/redis/releases
- **วิธีการที่ 2 : ใช้งาน Windows Subsystem for Linux (WSL) ของ Window ตามลิงค์ด้านล่าง**

  https://medium.com/@RedisLabs/windows-subsystem-for-linux-wsl-10e3ca4d434e
- **วิธีการที่ 3 : ใช้ Docker ตามลิงค์ด้านล่าง**

  https://hub.docker.com/_/redis
- **วิธีการที่ 4 : ใช้งาน Redis Cloud (free 30mb) ตามลิงค์ด้านล่าง**

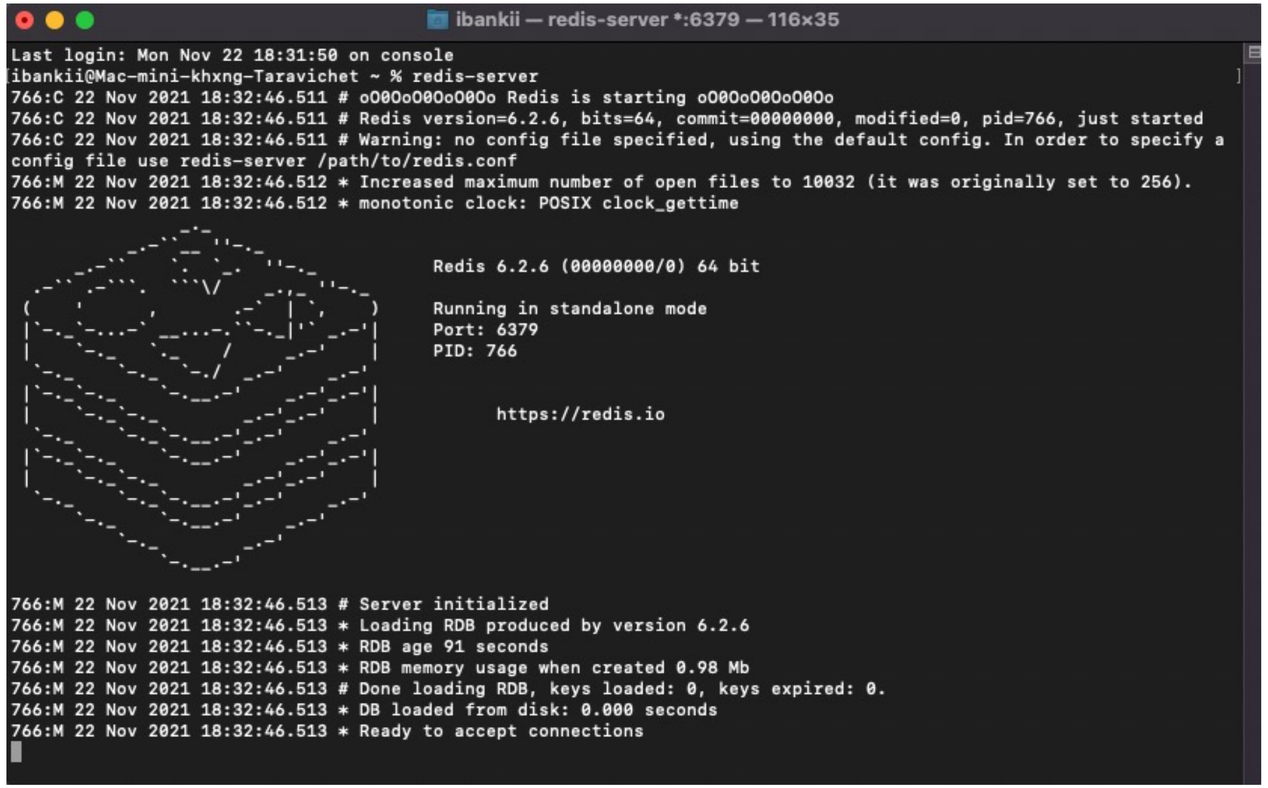
  https://app.redislabs.com/

---

**วิธีการติดตั้งสำหรับ MAC ผ่าน Terminal (ที่มี Brew)**

```
brew update
brew install redis
```

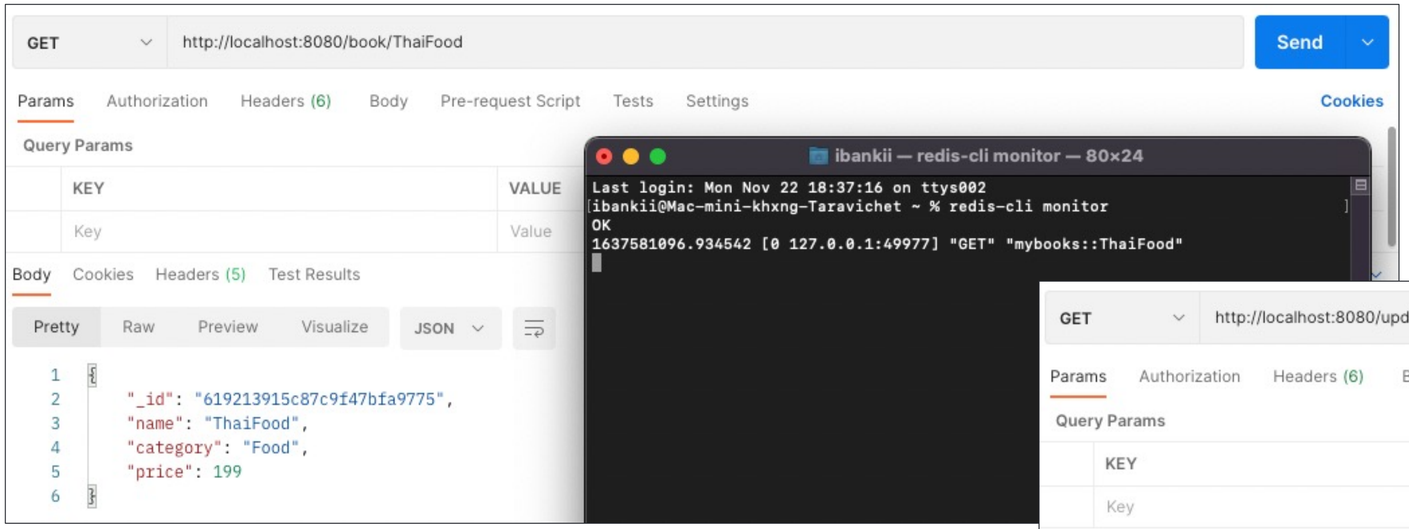
# วิธีการ Start ตัว Redis




- การสั่งให้ Redis เริ่มการทำงาน

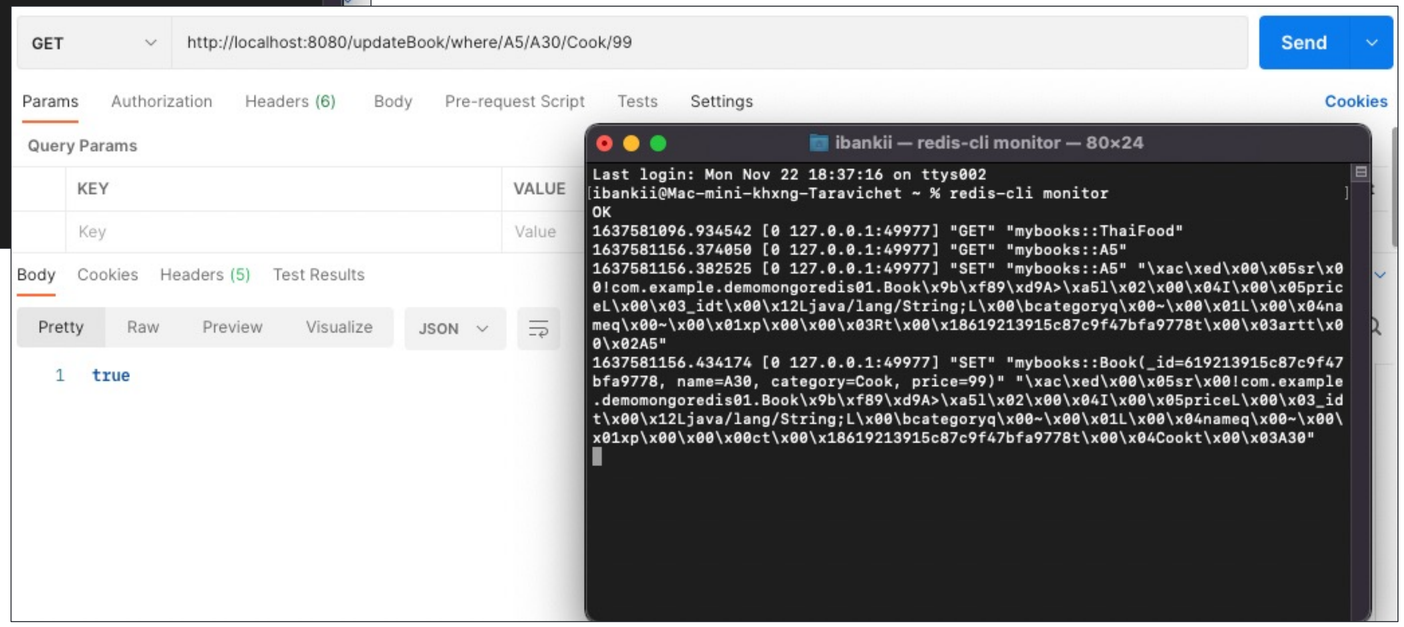
```
redis-server
```

# วิธีการ Monitoring ตัว Redis













- การสังเกตุการสื่อสารของ Redis

```
redis-cli monitor
```


# Outline

- แนวทางการออกแบบระบบด้วย Repository Pattern
- แนวคิดของฐานข้อมูลแบบ NoSQL และการใช้งานฐานข้อมูล MongoDB
- การโปรแกรม Spring Boot สำหรับเชื่อมต่อฐานข้อมูล MongoDB แบบ Repository Pattern
- การแคชข้อมูล (Caching) ด้วย Redis