

# Chapter 3

## Inter-Process Communication - JSON with REST

---

บรรยายโดย ผศ.ดร.ธราวดิษฐ์ ริติจรูณโรจน์ และอาจารย์สัญชัย น้อยจันทร์

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง



# Outline

- Communication in a monolithic architecture
- Dependency Injection
- Interaction Styles
  - Synchronous vs Asynchronous
  - Request – Response
- Message Format
- Communication Interface
  - Representational State Transfer (REST) ”
- Introduction to Spring Framework and Spring Boot



# Outline

- Communication in a monolithic architecture
- Dependency Injection
- Interaction Styles
  - Synchronous vs Asynchronous
  - Request – Response
- Message Format
- Communication Interface
  - Representational State Transfer (REST)”
- Introduction to Spring Framework and Spring Boot



# Communication in a monolithic architecture

ระบบ monolithic ที่ทำงานแบบกระบวนการเดียว (Single Process) จะมีการสื่อสารกันภายในระหว่างส่วนหรือองค์ประกอบโดยอาศัย

- 1. การเรียกใช้เมธอด
- 2. สร้างวัตถุ เช่น new Book()

สามารถเรียกใช้แบบ Dependency Non-Injection

```
public class App {  
    private DatabaseThingie myDatabase;  
    public App() {  
        myDatabase = new DatabaseThingie();  
    }  
    public void doSomething() {  
        myDatabase.getData();  
    }  
}
```

สิ่งเหล่านี้สามารถเชื่อมต่อกันได้อย่างมาก (Tightly Coupled) และ Object เหล่านี้จะทำงานอยู่บน Process เดียวกัน



# Outline

- Communication in a monolithic architecture
- **Dependency Injection**
- Interaction Styles
  - Synchronous vs Asynchronous
  - Request – Response
- Message Format
- Communication Interface
  - Representational State Transfer (REST)”
- Introduction to Spring Framework and Spring Boot



# Dependency Injection

```
public class App {  
    private DatabaseThingie myDatabase;  
    public App() {  
        myDatabase = new DatabaseThingie();  
    }  
    public void doSomething() { myDatabase.getData(); }  
}
```

ในคลาสใด ๆ ถ้าต้องเรียกใช้เมธอดของอีกคลาสนึงจะต้องทำการสร้าง Instance ของ Object ขึ้นมาก่อน โดยสิ่งนี้จะเรียกว่า “Dependency” หรือ “Instance Variable” จากตัวอย่าง Dependency Non-Injection จะพบว่าคลาส App และคลาส DatabaseThingie เกิดการผูกมัดกันมาก **เนื่องจากตัวแปร myDatabase หรือ Dependency ถูกสร้างใน Constructor ของคลาส App ซึ่งอาจก่อให้เกิดปัญหาต่าง ๆ ได้แก่**

- **ประจำ** : คลาส App จะไม่สามารถทำงานได้ถ้าคลาส DatabaseThingie เกิดการเปลี่ยนแปลง อาทิเช่น ต้องการพารามิเตอร์เพิ่มเติม
- **ไม่ยืดหยุ่น** : คลาส App ไม่รองรับการปรับเปลี่ยนชนิดซอฟท์แวร์ อาทิเช่น ต้องการเปลี่ยนจากคลาส DatabaseThingie ไปเป็นคลาส MockDatabase (เป็นคลาสลูกของ DatabaseThingie)
- **ยากต่อการทดสอบ** : สิ่งที่เกิดขึ้นในคลาส App อาจเป็นผลมาจากการ dependency ทำให้เราไม่สามารถควบคุมได้



# Dependency Injection

## Dependency Non-Injection

```
public class App {  
    private DatabaseThingie myDatabase;  
    public App() { myDatabase = new DatabaseThing(); }  
    public void doSomething() { myDatabase.getData(); }  
}
```

## Dependency Injection

```
public class App {  
    private DatabaseThingie myDatabase;  
    public App() { myDatabase = new DatabaseThing(); }  
    public App(DatabaseThing useThisDatabaseInstead) {  
        myDatabase = useThisDatabaseInstead;  
    }  
    public void doSomething() { myDatabase.getData(); }  
}
```



# Dependency Injection

```
public class AppTest {  
    public void testDoSomething() {  
        MockDatabase mockDatabase = new MockDatabase();  
        // MockDatabase is a subclass of DatabaseThing  
        // "inject" it here:  
        App app = new App(mockDatabase);  
        app.doSomething();  
    }  
}
```

```
public class App {  
    private DatabaseThing myDatabase;  
    public App() { myDatabase = new DatabaseThing(); }  
    public App(DatabaseThingie useThisDatabaseInstead) {  
        myDatabase = useThisDatabaseInstead;  
    }  
    public void doSomething() { myDatabase.getData(); }  
}
```

ดังนั้น นักศึกษาสามารถแก้ไขได้โดย

- สร้าง Constructor ที่รองรับพารามิเตอร์ DatabaseThing เพิ่ม
- กำหนดค่าแออทริบิวต์ผ่านเมธอด (setter)

สิ่งนี้ทำให้ห้องสอนคลาสลดการผูกมัดกันลงและทำให้สะดวกต่อการทดสอบการทำงานของแต่ละส่วน



# Outline

- Communication in a monolithic architecture
- Dependency Injection
- Interaction Styles
  - Synchronous vs Asynchronous
  - Request – Response
- Message Format
- Communication Interface
  - Representational State Transfer (REST)”
- Introduction to Spring Framework and Spring Boot



# Interaction Styles

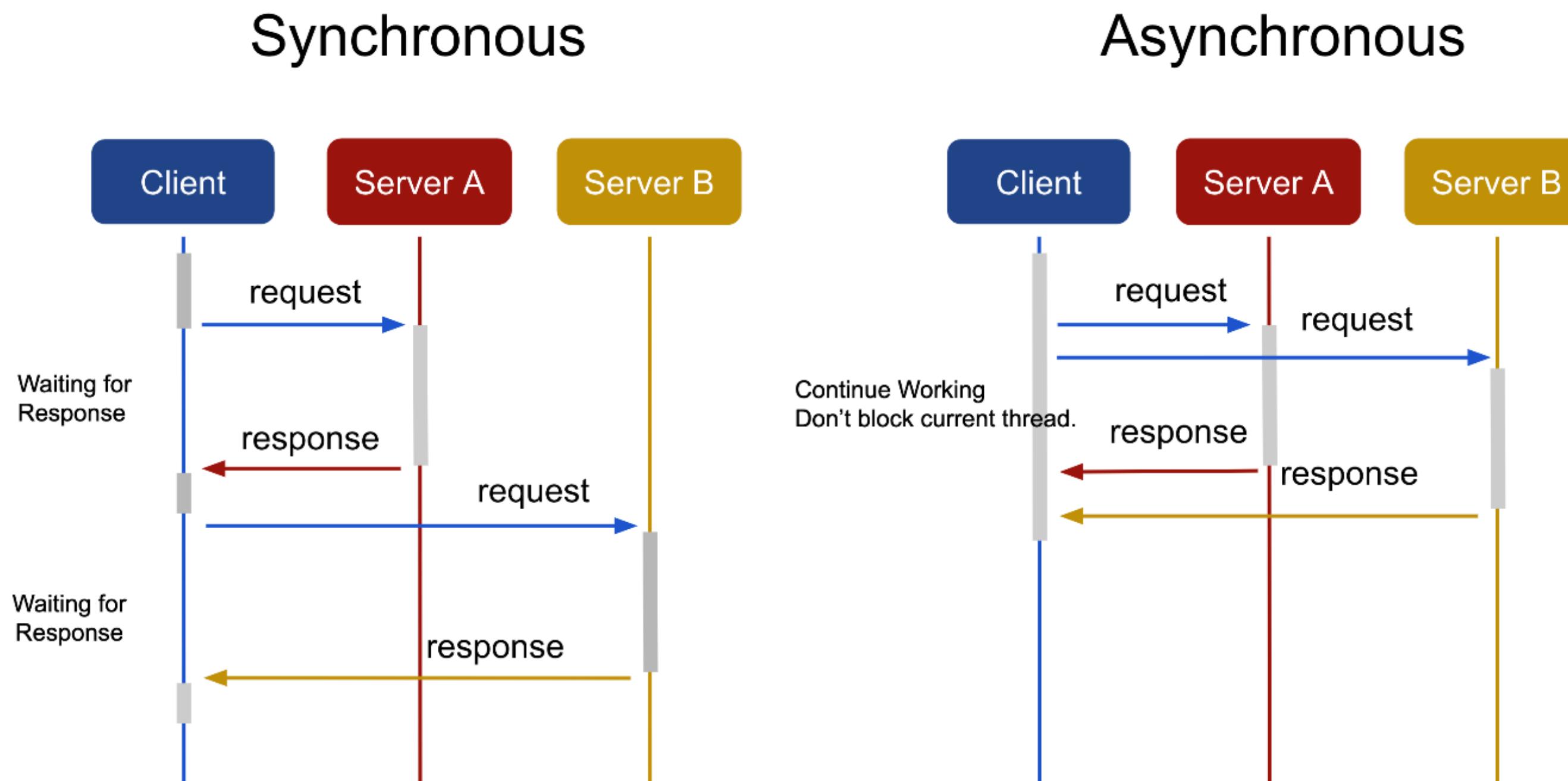
การเลือกวิธีการสื่อสารภายใน (IPC) นักศึกษาควรพิจารณารูปแบบการสื่อสารระหว่าง Service กับ Client ก่อน โดยควรพิจารณาจากความ **ต้องการ** และการ **ออกแบบของระบบ** เพื่อหลีกเลี่ยงการตกหล่นและได้วิธีการสื่อสารที่ตอบโจทย์ กับระบบที่ได้ออกแบบไว้ ซึ่งส่งผลต่อการพร้อมใช้งาน (Availability) ของระบบของนักศึกษา

การสื่อสารระหว่าง Server และ Client มีหลากหลายรูปแบบ ซึ่งสามารถแบ่งได้ตาม (1) จำนวนผู้รับสาร และ (2) โปรโตคอล (Protocol) ดังแสดงในตารางต่อไปนี้

	one-to-one	one-to-many
Synchronous	<ul style="list-style-type: none"><li>Request – Response</li></ul>	-
Asynchronous	<ul style="list-style-type: none"><li>Asynchronous Request - Response</li><li>Notifications (one-way request)</li></ul>	<ul style="list-style-type: none"><li>Publish – Subscribe</li><li>Publish – Asynchronous Response</li></ul>



# Synchronous vs Asynchronous



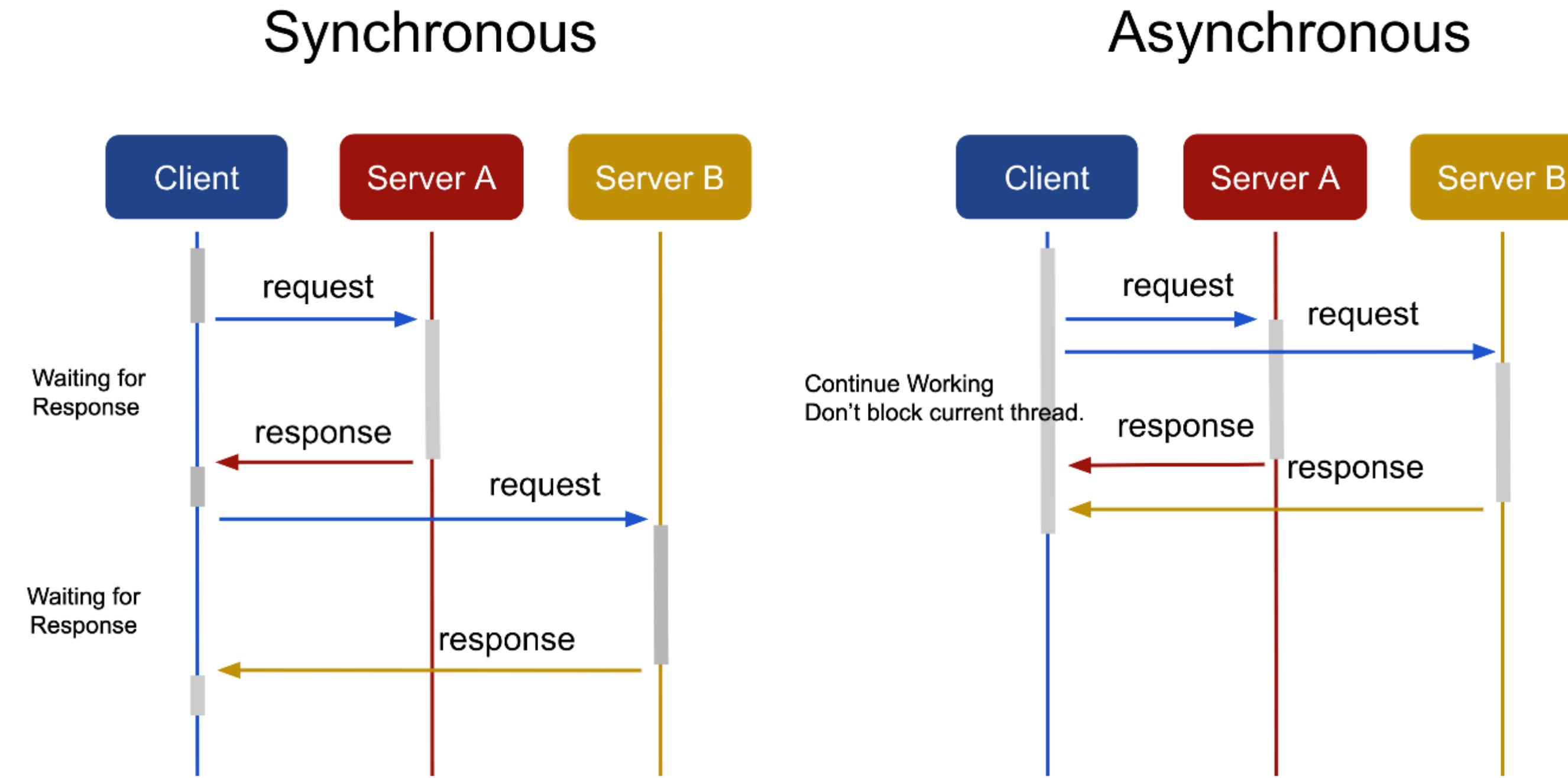
อ้างอิง <https://engineering.linecorp.com/en/blog/reactive-streams-arteria-1/>

โปรโตคอล **Synchronous** เช่น HTTP ที่เป็นการสื่อสารแบบพร้อมกัน โดยเริ่มจาก Client ร้องขอサービス (Request) บางอย่างจาก Server จนกว่าจะได้รับตอบกลับ (Response) จาก Server ถึง Client จะสามารถทำงานต่อได้

โปรโตคอล **Asynchronous** เช่น AMQP (เป็นโปรโตคอลที่รองรับในหลายระบบปฏิบัติการและคลาวน์) ที่เป็นการสื่อสารแบบไม่พร้อมกัน โดยเริ่มจาก Client ร้องขอบริการ (Request) บางอย่างจาก Server จนกว่าจะได้รับตอบกลับ (Response) จาก Server ทำให้ Client สามารถไปทำงานอื่น ๆ ต่อได้ ซึ่งคล้ายกับการส่ง Message ไปที่ Queue หรือ Message Broker



# Synchronous vs Asynchronous



อ้างอิง <https://engineering.linecorp.com/en/blog/reactive-streams-arma-1/>

## ข้อดีของวิธีการ Asynchronous ได้แก่

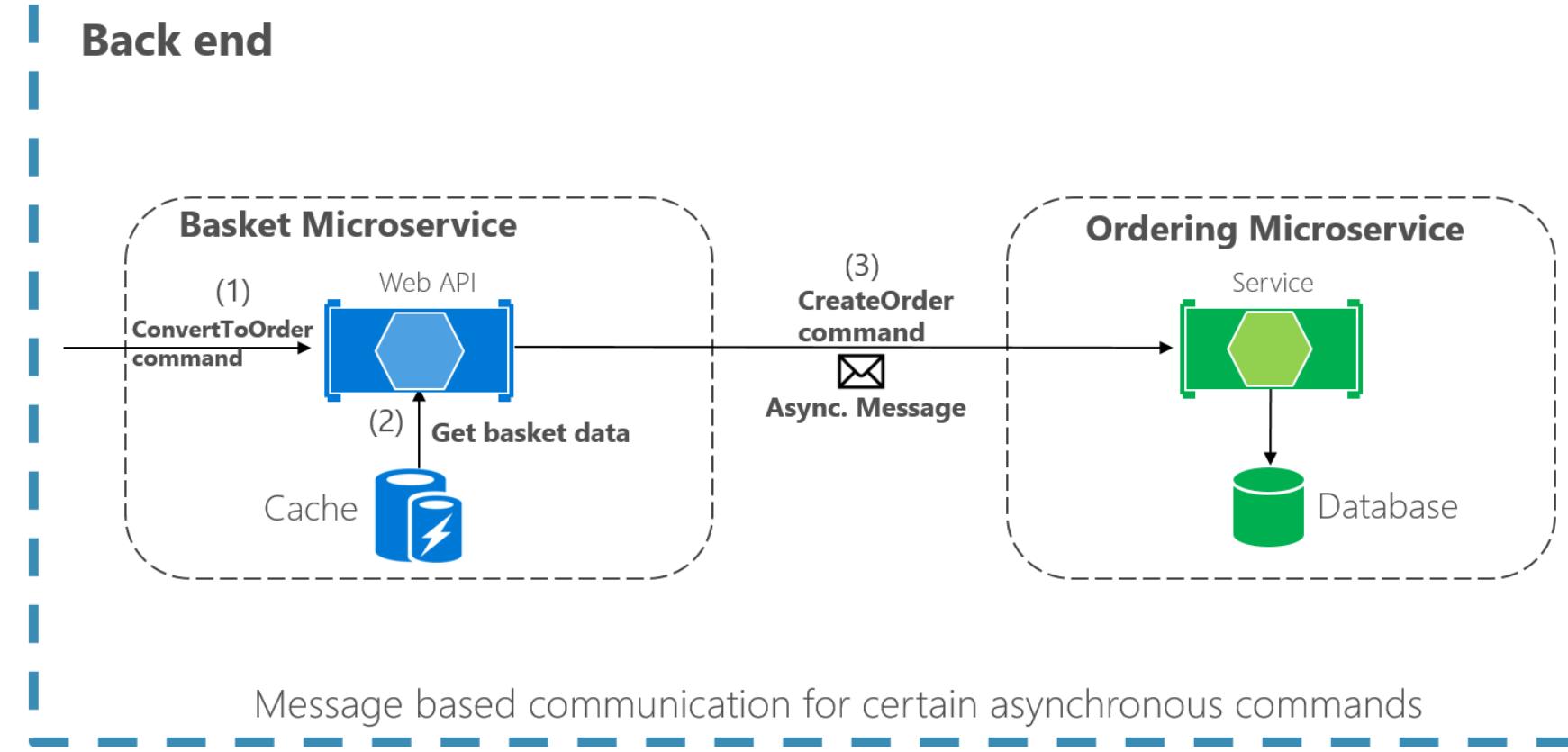
- มีความรวดเร็ว (Fast Speed) กล่าวคือ นักศึกษาสามารถได้รับ Response ทันทีหลังจากส่ง Request ได้เร็วขึ้น
- ใช้ทรัพยากรน้อย (Less Resource) กล่าวคือ นักศึกษาสามารถส่ง request ได้มากขึ้นโดยไม่ต้องใช้งานแพร์เด



# Single vs Multiple Receivers

## Single receiver message-based communication

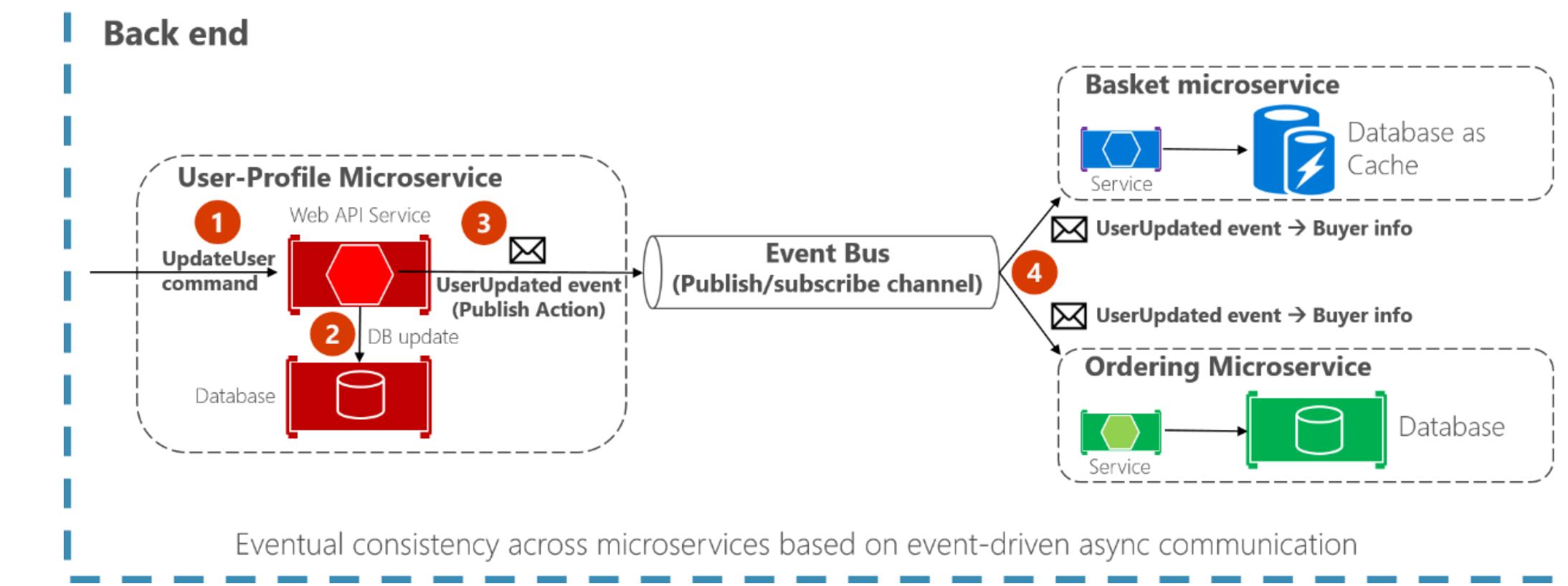
(i.e. Message-based Commands)



Message based communication for certain asynchronous commands

## Asynchronous event-driven communication

Multiple receivers



Eventual consistency across microservices based on event-driven async communication

อ้างอิง <https://docs.microsoft.com/th-th/dotnet/architecture/microservices/architect-microservice-container-applications/asynchronous-message-based-communication>

- **Single Receiver** คือ แต่ละ Request จะถูกส่งไป Process ที่ Service เพียง Service เดียว
- **Multiple Receiver** คือ แต่ละ Request จะถูกส่งไป Process ที่ Service ตั้งแต่ศูนย์ถึงหลายในทางปฏิบัติจะเป็นการสื่อสารโดยอาศัยプロTOCOLแบบ Asynchronous

โดยทั่วไปแล้วการพัฒนาระบบแบบ Microservice จะอาศัยรูปแบบการสื่อสารที่หลากหลายร่วมกัน แต่ส่วนใหญ่จะเป็นรูปแบบ Single Receiver ด้วย protocol Synchronous



# Interaction Styles

การเลือกวิธีการสื่อสารภายใน (IPC) นักศึกษาควรพิจารณารูปแบบการสื่อสารระหว่าง Service กับ Client ก่อน โดยควรพิจารณาจากความ **ต้องการ** และการ **ออกแบบของระบบ** เพื่อหลีกเลี่ยงการตกหล่นและได้วิธีการสื่อสารที่ตอบโจทย์ กับระบบที่ได้ออกแบบไว้ ซึ่งส่งผลต่อการพร้อมใช้งาน (Availability) ของระบบของนักศึกษา

การสื่อสารระหว่าง Server และ Client มีหลากหลายรูปแบบ ซึ่งสามารถแบ่งได้ตาม (1) จำนวนผู้รับสาร และ (2) โปรโตคอล (Protocol) ดังแสดงในตารางต่อไปนี้

	one-to-one	one-to-many
Synchronous	<ul style="list-style-type: none"><li>Request – Response</li></ul>	-
Asynchronous	<ul style="list-style-type: none"><li>Asynchronous Request - Response</li><li>Notifications (one-way request)</li></ul>	<ul style="list-style-type: none"><li>Publish – Subscribe</li><li>Publish – Asynchronous Response</li></ul>



# Interaction Styles

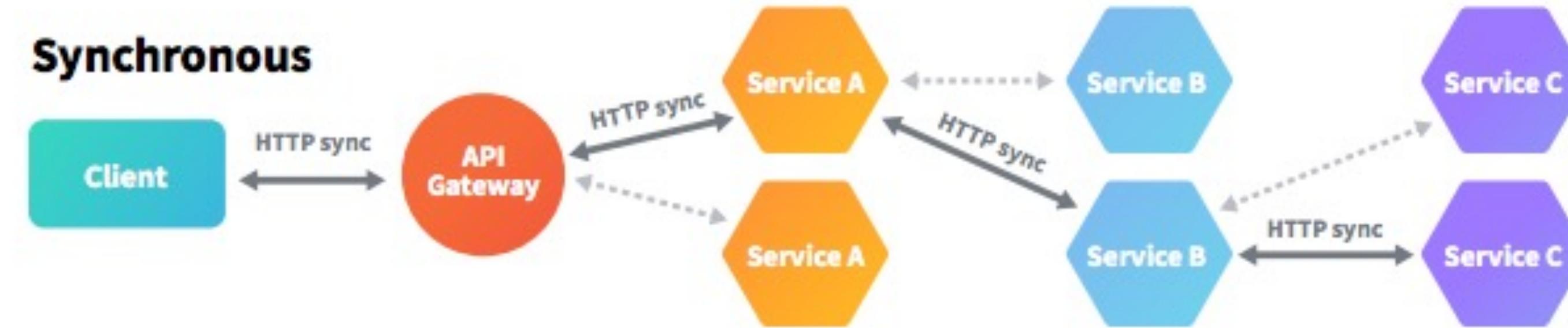
การเลือกวิธีการสื่อสารภายใน (IPC) นักศึกษาควรพิจารณารูปแบบการสื่อสารระหว่าง Service กับ Client ก่อน โดยควรพิจารณาจากความ ต้องการ และการ ออกแบบของระบบ เพื่อหลีกเลี่ยงการตกหล่นและได้วิธีการสื่อสารที่ตอบโจทย์ กับระบบที่ได้ออกแบบไว้ ซึ่งส่งผลต่อการพร้อมใช้งาน (Availability) ของระบบของนักศึกษา

การสื่อสารระหว่าง Server และ Client มีหลากหลายรูปแบบ ซึ่งสามารถแบ่งได้ตาม (1) จำนวนผู้รับสาร และ (2) โปรโตคอล (Protocol) ดังแสดงในตารางต่อไปนี้

	one-to-one	one-to-many
Synchronous	<ul style="list-style-type: none"><li>Request – Response</li></ul>	-
Asynchronous	<ul style="list-style-type: none"><li>Asynchronous Request - Response</li><li>Notifications (one-way request)</li></ul>	<ul style="list-style-type: none"><li>Publish – Subscribe</li><li>Publish – Asynchronous Response</li></ul>



# Request – Response



อ้างอิง <https://dzone.com/articles/communicating-between-microservices>

หลักการสื่อสารระหว่างผู้ร้องขอรับบริการ (Client หรือ Service) กับผู้ให้บริการ (Server หรือ Service) ผ่านรูปแบบ  
การสื่อสาร Request – Response มีขั้นตอนดังต่อไปนี้

ขั้นที่ 1 ร้องขอบริการ (Request) บางอย่าง จาก Client ส่งไปยัง Server

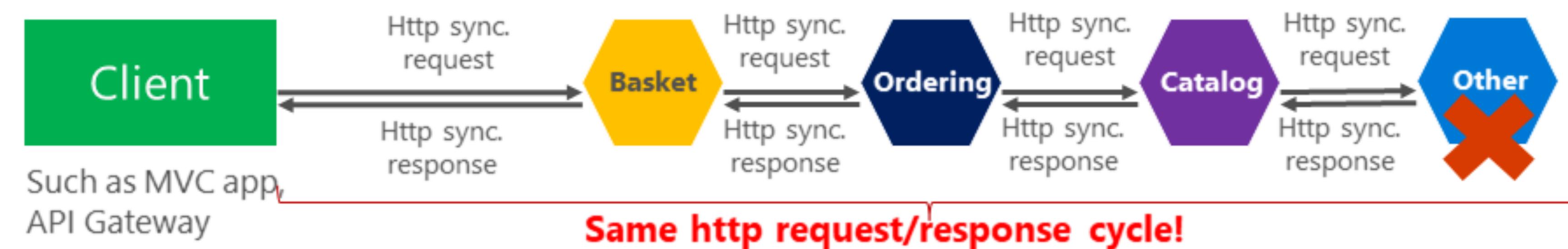
ขั้นที่ 2 รอการตอบกลับ (Response) จาก Server

ขั้นที่ 3 ส่งการตอบกลับไปที่ Client



# Request – Response

ในขั้นที่ 2 Thread ของ Client จะอยู่ในสถานะ Block (แต่ก็ขึ้นอยู่กับโค้ดของ Client ด้วย) แต่ด้วยลักษณะการสื่อสารแบบ Request – Response จะอาศัยโปรโตคอล HTTP หรือ HTTPS ที่มีการทำงานแบบ Synchronous ทำให้ Client ไม่สามารถทำงานต่อได้จนกว่าจะได้รับ Response จาก Server ตอบกลับมา สิ่งนี้ทำให้การสื่อสารแบบ Request – Response อาจก่อให้เกิด Tightly Coupled ระหว่าง Service ได้



อ้างอิง <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture>

จากภาพข้างต้น นักศึกษาจะพบว่าหนึ่ง Request อาจจะถูกส่งต่อไปยัง Service ที่แตกต่างกันมาก many ถ้าเกิดเหตุการณ์ที่มีหนึ่ง Service ไม่พร้อมให้บริการเนื่องจาก (1) ล่ม หรือ (2) บำรุงรักษา หรือ (3) โหลดเกิน หรือ (4) ตอบกลับล่าช้ามาก จากปัญหาข้างต้นสามารถแก้ไขได้โดย

- การตั้งค่าระบบเครือข่าย เกี่ยวกับ timeouts เพื่อหลีกเลี่ยงปัญหาตอบกลับล่าช้ามาก
- การจำกัดจำนวนหรืองดรับ Request ที่จะประมวลผล กรณีที่ตรวจสอบว่า Service ล่มหรือตอบสนองล่าช้า
- อาศัยหลักการ Circuit Breaker Pattern และ Fallback
- อาศัยหลักการ Tracing, Logging และ Monitoring



# Interaction Styles

การเลือกวิธีการสื่อสารภายใน (IPC) นักศึกษาควรพิจารณารูปแบบการสื่อสารระหว่าง Service กับ Client ก่อน โดยควรพิจารณาจากความ ต้องการ และการ ออกแบบของระบบ เพื่อหลีกเลี่ยงการตกหล่นและได้วิธีการสื่อสารที่ตอบโจทย์ กับระบบที่ได้ออกแบบไว้ ซึ่งส่งผลต่อการพร้อมใช้งาน (Availability) ของระบบของนักศึกษา

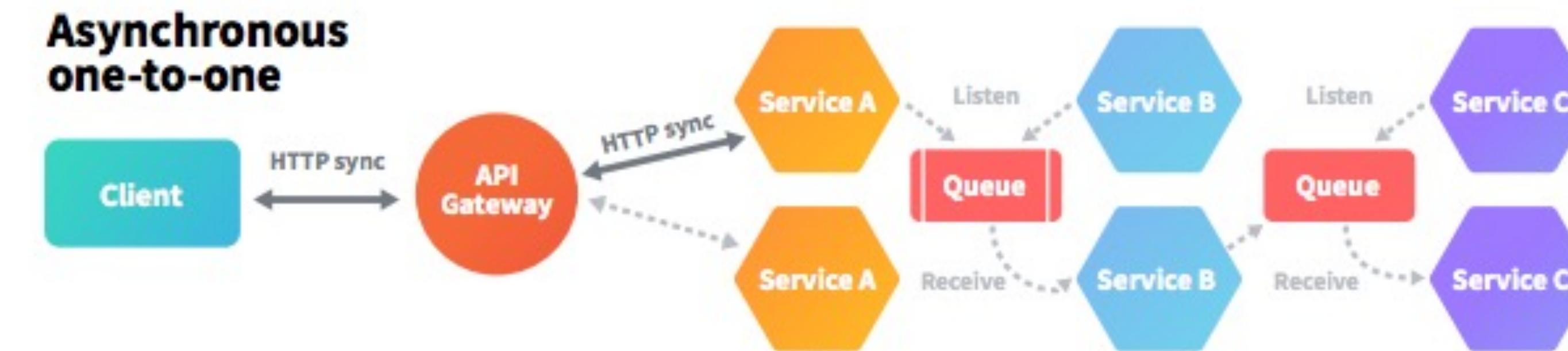
การสื่อสารระหว่าง Server และ Client มีหลากหลายรูปแบบ ซึ่งสามารถแบ่งได้ตาม (1) จำนวนผู้รับสาร และ (2) โปรโตคอล (Protocol) ดังแสดงในตารางต่อไปนี้

	one-to-one	one-to-many
Synchronous	<ul style="list-style-type: none"><li>Request – Response</li></ul>	-
Asynchronous	<ul style="list-style-type: none"><li>Asynchronous Request - Response</li><li>Notifications (one-way request)</li></ul>	<ul style="list-style-type: none"><li>Publish – Subscribe</li><li>Publish – Asynchronous Response</li></ul>



# Asynchronous Request - Response

หลักการสื่อสารระหว่างผู้ร้องขอรับบริการ (Client หรือ Service) กับผู้ให้บริการ (Server หรือ Service) ผ่านรูปแบบ การสื่อสาร Asynchronous Request – Response มีขั้นตอนดังต่อไปนี้



อ้างอิง <https://dzone.com/articles/communicating-between-microservices>

เมื่อ Client มีการส่ง Request ไปที่ Server ในรูปแบบ Asynchronous Request และจะเกิดการคืนการควบคุมให้ Client โดยไม่ต้องเสียเวลาการ Response จาก Server ทำให้สามารถประมวลหรือจัดการ Request อื่นได้ เป็นการทำให้เกิดกระบวนการ Decouple ระหว่าง Request และ Response ได้ Asynchronous Request – Response นิยมถูกใช้เมื่อต้องติดต่อกับ Service ที่มีค่า Latency มาก

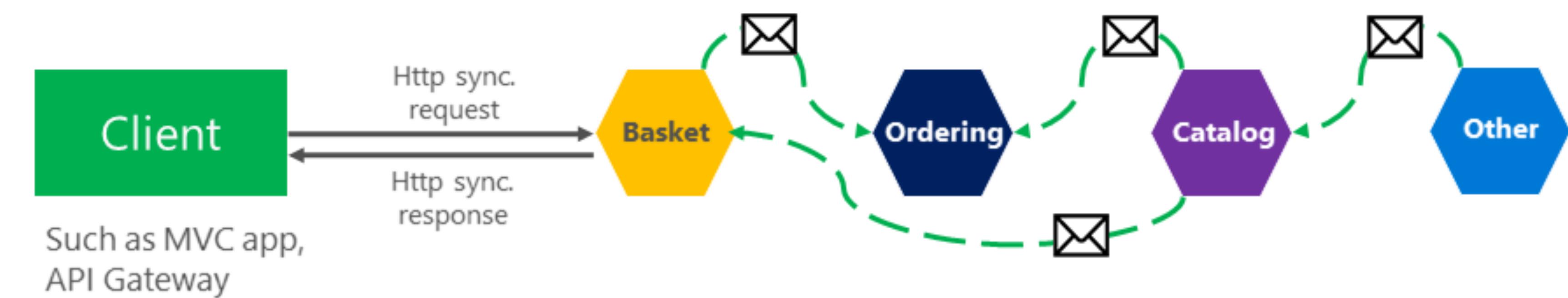


# Asynchronous Request - Response

Asynchronous Request – Response นิยมถูกใช้เมื่อต้องติดต่อกับ Service ที่มีค่า Latency มาก

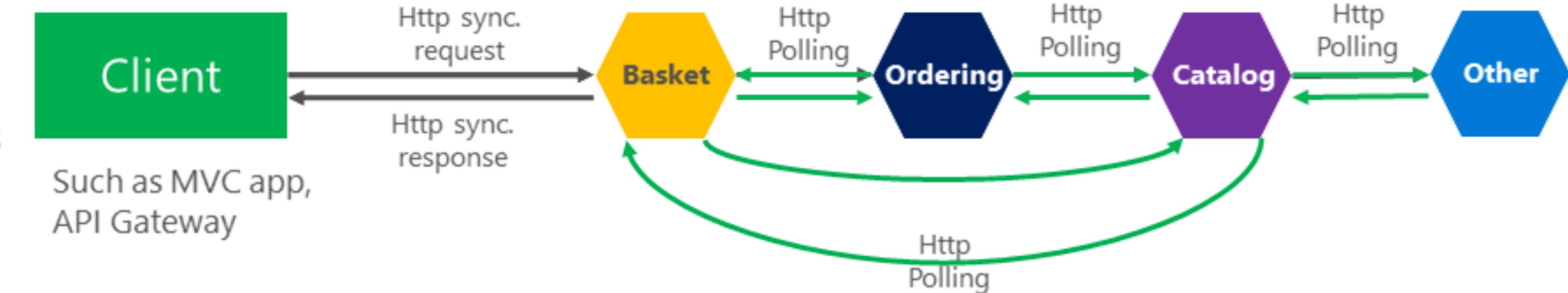
## Asynchronous

Comm. across internal microservices  
(EventBus: like **AMQP**)



## “Asynchronous”

Comm. across internal microservices  
(Polling: **Http**)



อ้างอิง <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture>



# Interaction Styles

การเลือกวิธีการสื่อสารภายใน (IPC) นักศึกษาควรพิจารณารูปแบบการสื่อสารระหว่าง Service กับ Client ก่อน โดยควรพิจารณาจากความ ต้องการ และการ ออกแบบของระบบ เพื่อหลีกเลี่ยงการตกหล่นและได้วิธีการสื่อสารที่ตอบโจทย์ กับระบบที่ได้ออกแบบไว้ ซึ่งส่งผลต่อการพร้อมใช้งาน (Availability) ของระบบของนักศึกษา

การสื่อสารระหว่าง Server และ Client มีหลากหลายรูปแบบ ซึ่งสามารถแบ่งได้ตาม (1) จำนวนผู้รับสาร และ (2) โปรโตคอล (Protocol) ดังแสดงในตารางต่อไปนี้

	one-to-one	one-to-many
Synchronous	<ul style="list-style-type: none"><li>Request – Response</li></ul>	-
Asynchronous	<ul style="list-style-type: none"><li>Asynchronous Request - Response</li><li>Notifications (one-way request)</li></ul>	<ul style="list-style-type: none"><li>Publish – Subscribe</li><li>Publish – Asynchronous Response</li></ul>



# Interaction Styles

การเลือกวิธีการสื่อสารภายใน (IPC) นักศึกษาควรพิจารณารูปแบบการสื่อสารระหว่าง Service กับ Client ก่อน โดยควรพิจารณาจากความ ต้องการ และการ ออกแบบของระบบ เพื่อหลีกเลี่ยงการตกหล่นและได้วิธีการสื่อสารที่ตอบโจทย์ กับระบบที่ได้ออกแบบไว้ ซึ่งส่งผลต่อการพร้อมใช้งาน (Availability) ของระบบของนักศึกษา

การสื่อสารระหว่าง Server และ Client มีหลากหลายรูปแบบ ซึ่งสามารถแบ่งได้ตาม (1) จำนวนผู้รับสาร และ (2) โปรโตคอล (Protocol) ดังแสดงในตารางต่อไปนี้

	one-to-one	one-to-many
Synchronous	<ul style="list-style-type: none"><li>Request – Response</li></ul>	-
Asynchronous	<ul style="list-style-type: none"><li>Asynchronous Request - Response</li><li>Notifications (one-way request)</li></ul>	<ul style="list-style-type: none"><li>Publish – Subscribe</li><li>Publish – Asynchronous Response</li></ul>



# Interaction Styles

การเลือกวิธีการสื่อสารภายใน (IPC) นักศึกษาควรพิจารณารูปแบบการสื่อสารระหว่าง Service กับ Client ก่อน โดยควรพิจารณาจากความ ต้องการ และการ ออกแบบของระบบ เพื่อหลีกเลี่ยงการตกหล่นและได้วิธีการสื่อสารที่ตอบโจทย์ กับระบบที่ได้ออกแบบไว้ ซึ่งส่งผลต่อการพร้อมใช้งาน (Availability) ของระบบของนักศึกษา

การสื่อสารระหว่าง Server และ Client มีหลากหลายรูปแบบ ซึ่งสามารถแบ่งได้ตาม (1) จำนวนผู้รับสาร และ (2) โปรโตคอล (Protocol) ดังแสดงในตารางต่อไปนี้

	one-to-one	one-to-many
Synchronous	<ul style="list-style-type: none"><li>Request – Response</li></ul>	-
Asynchronous	<ul style="list-style-type: none"><li>Asynchronous Request - Response</li><li>Notifications (one-way request)</li></ul>	<ul style="list-style-type: none"><li>Publish – Subscribe</li><li>Publish – Asynchronous Response</li></ul>



# Overall IPC

โครงสร้างการสื่อสารระหว่าง Service จะประกอบไปด้วยกัน 3 ส่วน ได้แก่ (1) Service, (2) Interface และ (3) Message





# Overall IPC

โครงสร้างการสื่อสารระหว่าง Service จะประกอบไปด้วยกัน 3 ส่วน ได้แก่ (1) Service, (2) Interface และ (3) Message





# Outline

- Communication in a monolithic architecture
- Dependency Injection
- Interaction Styles
  - Synchronous vs Asynchronous
  - Request – Response
- **Message Format**
- Communication Interface
  - Representational State Transfer (REST)”
- Introduction to Spring Framework and Spring Boot



# Message Format

ส่วนสำคัญของการสื่อสารแบบ IPC คือ กระบวนการแลกเปลี่ยนข้อมูลหรือสารสนเทศ **ผ่านข้อความ (Message)** โดยโครงสร้างของ Message จะส่งผลกระทบโดยตรงกับประสิทธิภาพการทำงานของ IPC ในทางปฏิบัติ

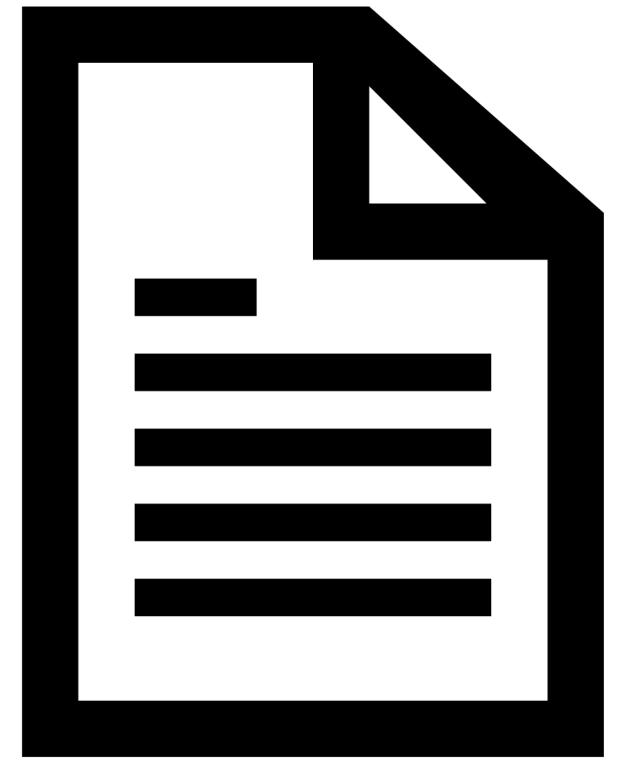
- แต่ละ Service ที่พัฒนาด้วย **ภาษาเดียวกัน** **ไม่**จำเป็นต้องใช้ Message ที่มีโครงสร้างเหมือนกัน
- แต่ละ Service ที่พัฒนาด้วย **ภาษาที่แตกต่างกัน** เดียวกัน อาจจะใช้ Message ที่มี **โครงสร้างเหมือนกัน**

แต่นักศึกษาไม่ควรใช้ Format ที่จำเพาะกับภาษา ออาทิเช่น JAVA Serialization เพราะจะทำให้ขาดความยืนหยุ่นและเป็นการผูกติดกับเทคโนโลยีที่ใช้พัฒนามากเกินไป



# Message Format

โครงสร้างของ Message จะประกอบด้วย 2 กลุ่มหลัก ได้แก่



เลขฐานสอง (Binary)

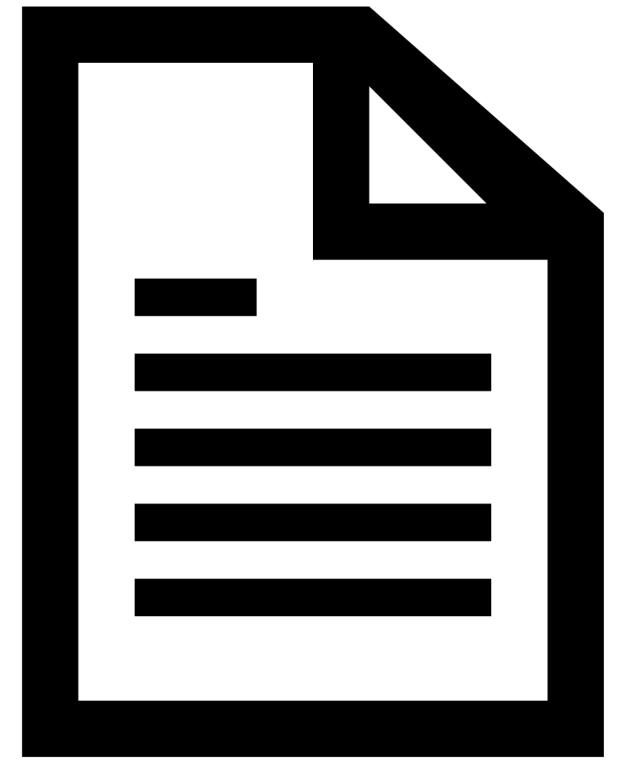


ตัวอักษร (Text)



# Message Format

โครงสร้างของ Message จะประกอบด้วย 2 กลุ่มหลัก ได้แก่



เลขฐานสอง (Binary)



ตัวอักษร (Text)



# Binary-based Message

คือ Message ประเภทเลขฐานสอง อាជิเช่น Protocol Buffers (Protobuf) และ AVRO โดยจะอาศัย Binary Protocol **Encoding** (หรือ Serialization) สำหรับเข้ารหัสข้อมูลให้อยู่ในรูปแบบเลขฐานสองก่อนนำส่งและ Binary Protocol **Decoding** (หรือ Deserialization) เพื่อถอดรหัสข้อมูลให้อยู่ในรูปแบบที่กำหนด (หรือข้อความ) ก่อนนำไปใช้งานผ่าน Complier โดยที่ Protocol Buffers และ AVRO ได้อาศัย Interactive Data Language (IDL) สำหรับกำหนดโครงสร้างของ Message นอกจากนี้ Protocol Buffers ได้อาศัย Tagged Fields ต่าง ๆ ขณะที่ AVRO ได้อาศัย Schema เพื่อทำความเข้าใจข้อมูลใน Message



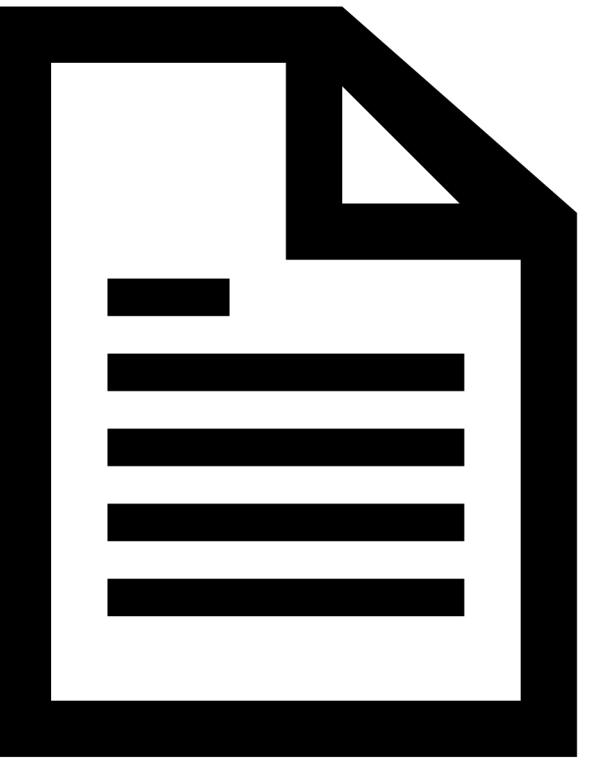


# Message Format

โครงสร้างของ Message จะประกอบด้วย 2 กลุ่มหลัก ได้แก่



เลขฐานสอง (Binary)



ตัวอักษร (Text)



# Text-based Message

คือ Message ประเภทตัวอักษรที่ง่ายต่อการและทำความเข้าใจของมนุษย์ ได้แก่ JSON และ XML เป็นต้น

JavaScript Object Notation: **JSON** คือ การเก็บข้อมูลแบบกลุ่มของคุณลักษณะ

```
{
  [
    {
      "firstName": "John", "lastName": "Doe" },
    {
      "firstName": "Peter", "lastName": "Jones" }
  ]
}
```

Extensible Markup Language: **XML** คือ การเก็บข้อมูลแบบกลุ่มของ key-value แบบ Markup

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```



# Text-based Message

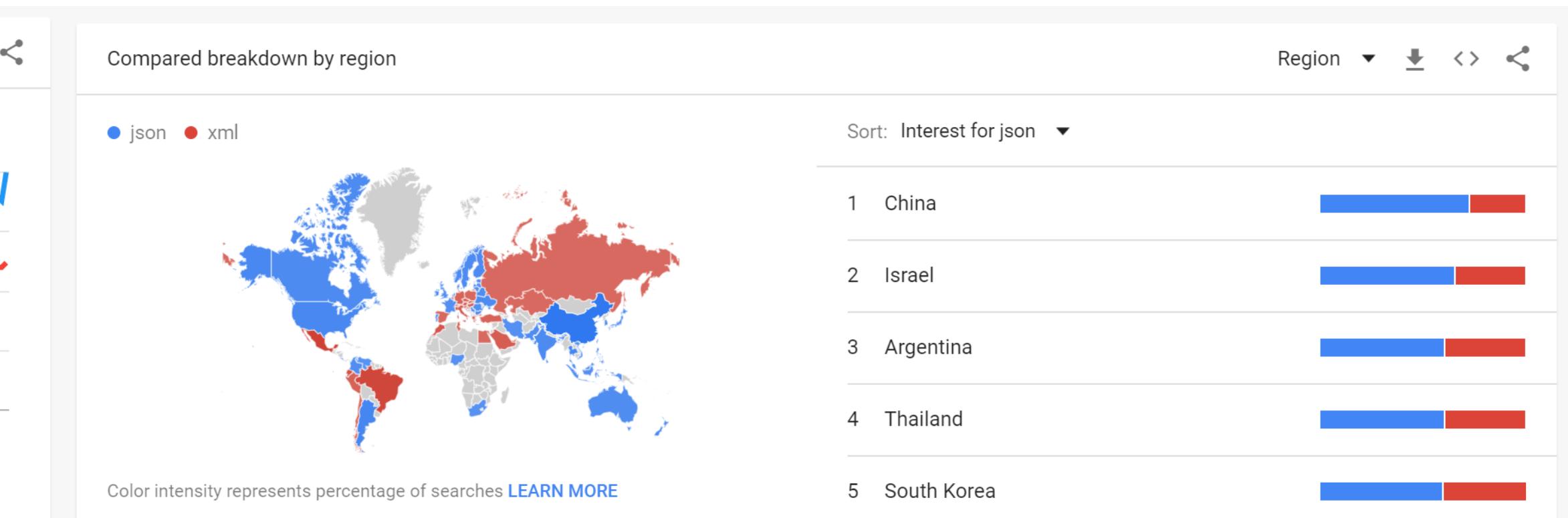
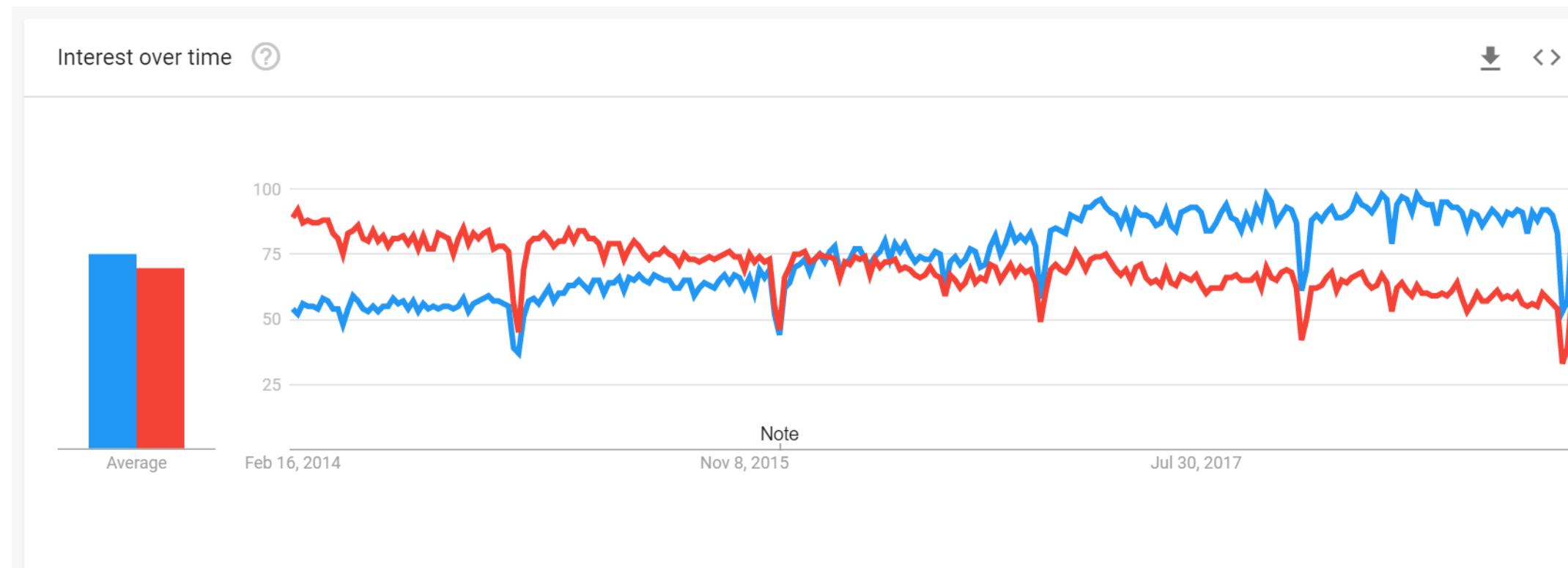
JavaScript Object Notation: **JSON**

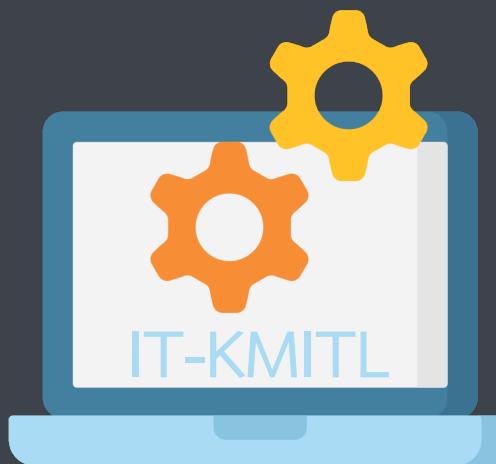
```
{  
  [  
    { "firstName": "John", "lastName": "Doe" },  
    { "firstName": "Peter", "lastName": "Jones" }  
  ]  
}
```

Extensible Markup Language: **XML**

```
<employees>  
  <employee>  
    <firstName>John</firstName> <lastName>Doe</lastName>  
  </employee>  
  <employee>  
    <firstName>Peter</firstName> <lastName>Jones</lastName>  
  </employee>  
</employees>
```

JSON และ XML คือ รูปแบบ (Format) สำหรับการรับส่งข้อมูลระหว่างเครื่องผู้ให้บริการ (Server) กับเครื่องผู้รับบริการ (Client) ซึ่งเป็นการคุยกันระหว่างแอพพลิเคชัน





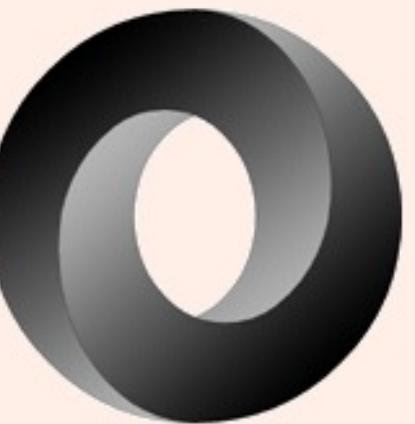
# Text-based Message

## JSON VERSUS XML

<p>JSON stands for JavaScript Object Notation and is based on JavaScript language.</p>	<p>XML is short for Extensive Markup Language and is derived from SGML.</p>
<p>It supports text and number data types including integer and strings, and arrays and objects.</p>	<p>It has no direct support for array.</p>
<p>It's a language-independent data-interchange format which supports only UTF-8 encoding.</p>	<p>It's an independent data format which supports different encodings.</p>
<p>It does not contain start and end tags.</p>	<p>It contains start and end tags.</p>
<p>It does not support native objects.</p>	<p>It gets support of objects via attributes and elements.</p>
<p>No support for Namespaces.</p>	<p>Namespaces are supported in XML.</p>



# JavaScript Object Notation



## Introducing JSON

العربية Български 中文 Český Dansk Nederlands English Esperanto Français Deutsch Ελληνικά עברית Magyar Indonesia Italiano 日本 한국어 فارسی Polski Português Română Русский Српско-хрватски Slovenčina Español Svenska Türkçe Tiếng Việt

ECMA-404 The JSON Data Interchange Standard.

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the [JavaScript Programming Language](#), [Standard ECMA-262 3rd Edition - December 1999](#). JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

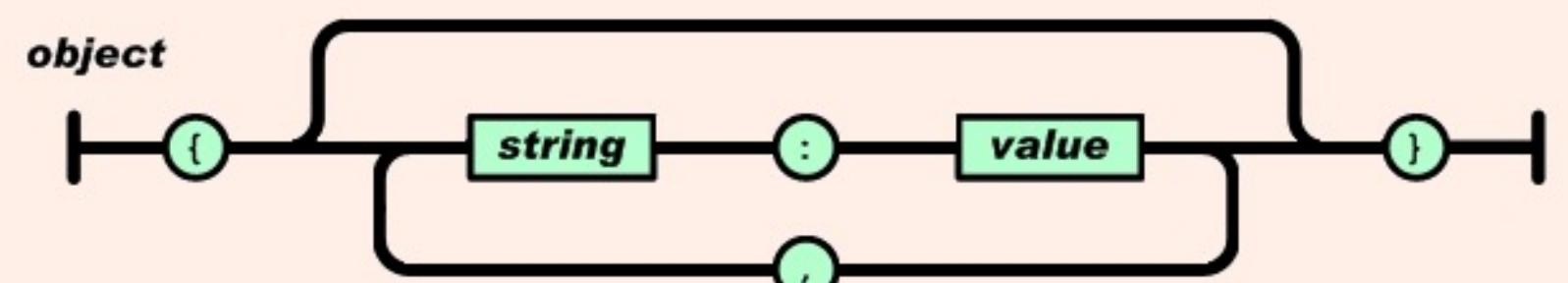
JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
  - An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).



```

object
  {}
  { members }

members
  pair
  pair , members

pair
  string : value

array
  []
  [ elements ]

elements
  value
  value , elements

value
  string
  number
  object
  array
  true
  false
  null

```

---

- พัฒนามาจากภาษา JavaScript
  - Douglas Crockford เป็นผู้คิดค้น JSON





# JavaScript Object Notation

โครงสร้างหลักของ JSON ประกอบด้วย 2 โครงสร้างย่อย ได้แก่

- **Object** เป็นการเก็บข้อมูลภายใต้เครื่องหมายเป็น {} แบบไม่มีลำดับในรูปแบบ Key – Value โดยอาศัยเครื่องหมาย Colon (:) สำหรับการแบ่งแยก Key และ Value นอกจากนี้ ยังอาศัยเครื่องหมาย Comma ( , ) มาช่วยแบ่งกรณีมี Key – Value มากกว่า 1 คู่ ตัวอย่างเช่น

```
{ "firstName": "John", "lastName": "Doe" }
```

- **Array** เป็นการเก็บข้อมูลภายใต้เครื่องหมายก้ามปุ [...] แบบมีลำดับในรูปแบบ Value โดยอาศัยเครื่องหมาย Comma ( , ) มาแบ่งแยกระหว่าง Value ตัวอย่างเช่น

```
[ "html", "xml", "css" ]
```



# JavaScript Object Notation

สำหรับ Value ของโครงสร้าง Object หรือ Array ใน JSON สามารถมีชนิดข้อมูลได้หลากหลาย ได้แก่

- ข้อความ (string) อาศัยเครื่องหมาย “....” เป็นตัวกำหนดขอบเขตเปิด-ปิด
- ตัวเลข (number) เป็นเลขจำนวนเต็ม, ทศนิยม หรือ Exponential (E) notation ในระบบเลขฐาน 10
- ค่าความจริง (boolean) เป็นค่า true หรือ false
- ค่าว่าง (null)
- Object เป็นโครงสร้างที่รองรับการซ้อนกันได้
- Array เป็นโครงสร้างที่รองรับการซ้อนกันได้

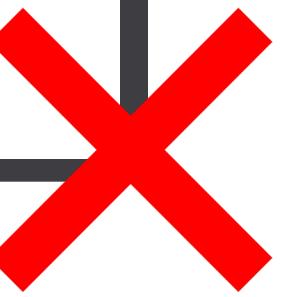
```
Human = {
    "name": 'Bank',           // String
    "age": 28,                // int
    "college" : true,         // boolean
    "offices" : [ '3350DMC', '3437NQ' ],   // List/Array
    "skills" : { "java": 10,
                 "C": 10,
                 "php": 5,
                 "python" : '7'
               }
};
```



# JavaScript Object Notation

## Example 1

```
{"skill": {"web": [{ "name": "html", "years": 5}, { "name": "css", "years": 3}], "database": [{ "name": "sql", "years": 7}]}}
```



## Example 2

```
{
  "skill": {
    "web": [
      { "name": "html", "years": 5},
      { "name": "css", "years": 3}
    ],
    "database": [
      { "name": "sql", "years": 7}
    ]
  }
}
```





# Overall IPC

โครงสร้างการสื่อสารระหว่าง Service จะประกอบไปด้วยกัน 3 ส่วน ได้แก่ (1) Service, (2) Interface และ (3) Message





# Outline

- Communication in a monolithic architecture
- Dependency Injection
- Interaction Styles
  - Synchronous vs Asynchronous
  - Request – Response
- Message Format
- **Communication Interface**
  - Representational State Transfer (REST)”
- Introduction to Spring Framework and Spring Boot



# Representational State Transfer (REST)



ในปี ค.ศ. 2000 Roy Fielding ได้คิดค้นสถาปัตยกรรมที่ใช้พัฒนา Web Service ขึ้นที่เรียกว่า “Representational State Transfer (REST)” โดยอาศัยโปรโตคอล Hypertext Transfer Protocol (HTTP) ของเว็บไซต์

หลักการของ REST จะให้ความสำคัญกับ Business Object (เช่น Order Service) ไม่ว่าจะเป็นแบบเดียว (Single) หรือกลุ่ม (Collection) โดยเชื่อมต่อกับ URL ผ่านโปรโตคอล HTTP ตัวอย่างเช่น Service A สร้าง Order Object โดยติดต่อกับ Order Service ด้วยวิธีการ POST และ Service A เรียกดูข้อมูลจาก Order Object ในรูปแบบ Text-based Message ด้วยวิธีการ GET ผ่าน **/orders/{orderId}**

จากตัวอย่างข้างต้นจะพบว่า REST จะมองว่า GET ได้รับการออกแบบมาสำหรับ Retrieve ข้อมูล ขณะที่ POST ได้รับการออกแบบมาสำหรับ จัดการ (สร้าง) Business Object นอกจากนี้ ทรัพยากรที่เรียกใช้บริการ ได้แก่ รูป, วีดีโอ, เว็บเพจ หรือข้อมูลทางธุรกิจ



# รูปแบบ HTTP Request

HTTP Request สำหรับ REST จะประกอบไปด้วย 5 ส่วนหลัก ได้แก่

## VERB

กำหนดตัวดำเนินการ อาทิเช่น GET

## URI

กำหนดที่อยู่ของ Resource

## HTTP Version

ส่วนมากจะกำหนดเป็น *HTTP v1.1*

## Request Header

กำหนด metadata ต่าง ๆ รวมถึง format ของข้อมูลใน Request Body

## Request Body

กำหนดข้อมูลของ REST



# ตัวอย่างรูปแบบ HTTP Request

## HTTP Request ในรูปแบบ POST Method

```
POST http://MyService/Person/  
Host: MyService  
Content-Type: text/xml; charset=utf-8  
Content-Length: 123  
<?xml version="1.0" encoding="utf-8"?>  
<Person>  
    <Name>M Vaqqas</Name>  
    <Email>m.vaqqas@gmail.com</Email>  
</Person>
```

## HTTP Request ในรูปแบบ GET Method

```
GET http://www.w3.org/Protocols/rfc2616/rfc2616.html HTTP/1.1  
Host: www.w3.org  
Accept: text/html,application/xhtml+xml,application/xml; ...  
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64)  
AppleWebKit/537.36 ...  
Accept-Encoding: gzip,deflate,sdch  
Accept-Language: en-US,en;q=0.8,hi;q=0.6
```



# รูปแบบ HTTP Response

HTTP Response สำหรับ REST จะประกอบไปด้วย 4 ส่วนหลัก ได้แก่

## HTTP Version

ส่วนมากจะกำหนดเป็น *HTTP v1.1*

## Response Code

เลขสถานะการทำงาน 3 หลัก อาทิเช่น 200 OK

## Response Header

กำหนด metadata ต่าง ๆ รวมถึง format ของข้อมูลใน Request Body

## Response Body

กำหนดข้อมูลของ REST



# ตัวอย่างรูปแบบ HTTP Response

## HTTP Response ในรูปแบบ **GET Method**

```
HTTP/1.1 200 OK
Date: Sat, 23 Aug 2014 18:31:04 GMT
Server: Apache/2
Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT
Accept-Ranges: bytes
Content-Length: 32859
Cache-Control: max-age=21600, must-revalidate
Expires: Sun, 24 Aug 2014 00:31:04 GMT
Content-Type: text/html; charset=iso-8859-1
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml'>
<head><title>Hypertext Transfer Protocol --
HTTP/1.1</title></head>
<body>
...

```



# REST Operators

ตัวดำเนินการของ REST จะสอดคล้องกับ CRUD ดังนี้

CRUD Operation	Create	Read	Update	Delete
REST Operation	POST (สร้างข้อมูล)	GET (เรียกข้อมูลจาก URI ที่กำหนด)	PUT (แก้ไขข้อมูล)	DELETE (ลบข้อมูล)

จากตารางเป็นแค่แนวทางไม่ใช้ข้อบังคับ ทั้งนี้ ขึ้นอยู่กับการออกแบบหรือแล้วแต่ตกลงกันในทีมผู้พัฒนา ตัวอย่างเช่น เราสามารถออกแบบการส่งค่าผ่านคิววิ่ง string ผ่านตัวดำเนินการ POST และตัวดำเนินการ GET ได้



# ข้อกำหนดของสถาปัตยกรรม REST

ประกอบไปด้วย 7 ประการได้แก่

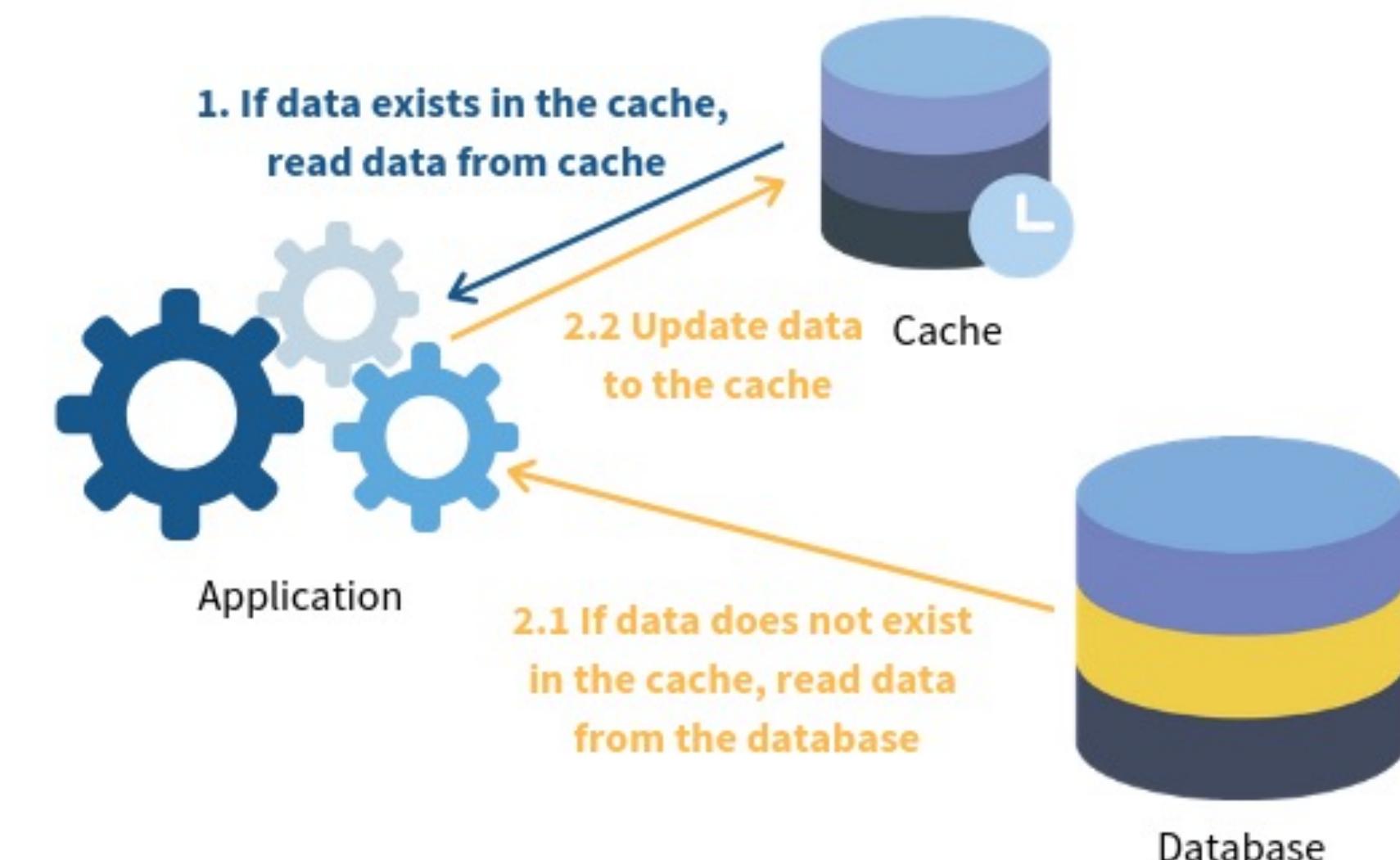
• Performance	มีประสิทธิภาพในการโต้ตอบกับ Request ที่เข้ามา
• Scalability	รองรับการปรับขยายขนาดเพื่อให้ง่ายต่อการเพิ่มลดส่วนประกอบต่าง ๆ
• Simplicity	มี Interface ที่ Simple และเหมือนกัน
• Modifiability	รองรับการปรับเปลี่ยนต่าง ๆ หรือเพิ่มลดส่วนประกอบตามความต้องการ ถึงแม้ว่าระบบกำลังทำงานอยู่
• Visibility	การมองเห็นการสื่อสารระหว่าง Component ผ่าน Service Agents
• Portability	ง่ายต่อการโยกย้ายในส่วนของ Program และ Data
• Reliability	มีความน่าเชื่อถือในการต้านทานความล้มเหลวในระบบ



# มาตรฐานของ RESTful API

เพื่อให้ง่ายต่อการพัฒนา และเป็นที่ยอมรับสำหรับการพัฒนา RESTful API ดังนั้น จึงเกิดมาตรฐานขึ้น 6 ข้อ ได้แก่

- **Client-server architecture:** Client ไม่จำเป็นต้องรู้อะไรเกี่ยวกับ Business logic ภายใน ไม่มีหน้าที่เกี่ยวกับการจัดเก็บข้อมูล ส่วน Server มีหน้าที่เก็บ Resource และไม่จำเป็นต้องรู้อะไรเกี่ยวกับ UI Frontend หรือสถานะของผู้เรียก
- **Cache-ability:** สามารถกำหนดประเภทของข้อมูลที่จะ Response ให้ Client ได้ว่าจะตอบเป็น (1) ข้อมูลเก่าที่อยู่ใน Cache หรือ (2) ข้อมูลปัจจุบัน





# มาตรฐานของ RESTful API

**Statelessness:** นักศึกษาควรเข้าใจก่อนว่า REST ย่อมาจาก Representational “**State**” Transfer โดยลักษณะของ Statelessness คือ การที่ Server ไม่ได้จัดเก็บสถานะต่าง ๆ ของ Client Session เอ้าไว้ที่ฝั่ง Server นอกจากนี้ ทุก Request จาก Client ที่ร้องขอไปยัง Server จะมีข้อมูลที่จำเป็นสำหรับการขอรับบริการนั้น ๆ เพียงเท่านั้น หมายความว่า Session state จะเก็บอยู่ฝั่ง Client เท่านั้น

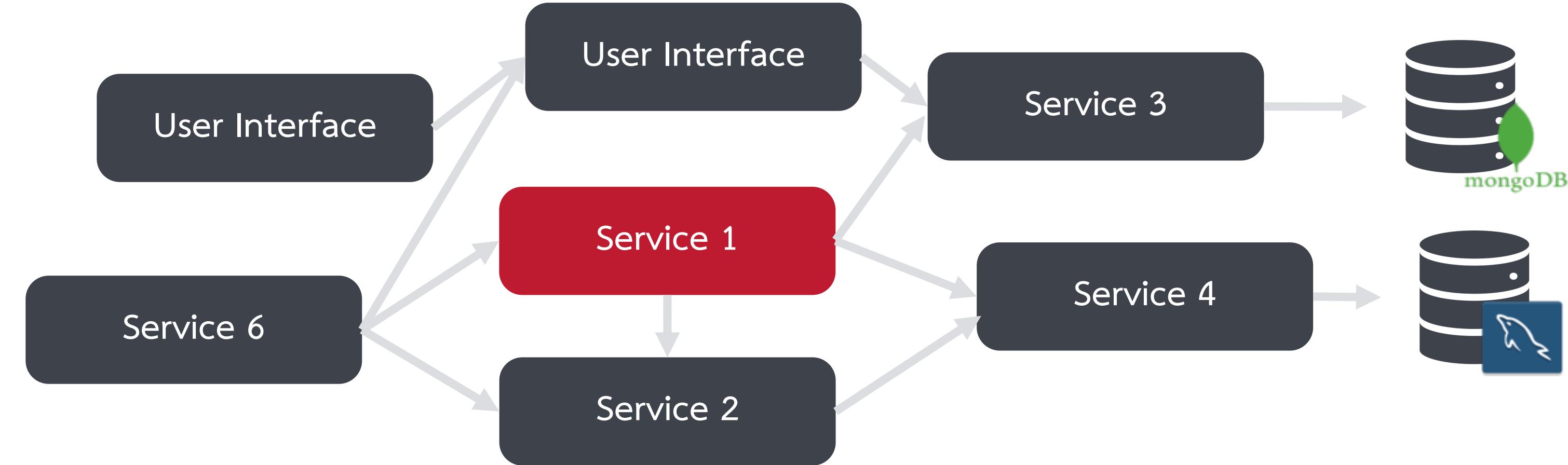
*Statelessness means that every HTTP request happens in complete isolation. When the client makes an HTTP request, it includes all information necessary for the server to fulfill that request. The server never relies on information from previous requests. If that information was important, the client would have sent it again in this request.*

Statelessness สำหรับ REST APIs คือ Application State ที่หมายความว่า ข้อมูลฝั่ง Server ที่จัดเก็บเพื่อ (1) ระบุตัวตนของ Client ที่ Request มา (2) รายละเอียดในการติดต่อกันหน้าแล้ว (3) ข้อมูลปัจจุบัน ซึ่งไม่ใช้ Resource State คือ สถานะปัจจุบันของ Resource บนเครื่อง Server ณ ช่วงเวลาหนึ่ง

“ส่ง Request รับ Response แล้วจบ ไม่มีการจัดเก็บสถานะต่าง ๆ ของ Client Session ไว้”



# มาตรฐานของ RESTful API



## Layered system

รองรับการทำงานแบบเป็นลำดับชั้นแบบเลยอร์ (Hierarchical layers) โดยทั่วไปหนึ่งเลยอร์จะประกอบด้วยองค์ประกอบที่มีการทำงานแตกต่างกันทำงานร่วมกัน และสามารถสื่อสารกับเลยอร์ด้านบนหรือเลยอร์ด้านล่างเท่านั้น เพื่อที่เลยอร์ด้านบนใช้เลยอร์ด้านล่างเรียกดู เพิ่ม ลบ และแก้ไข ซึ่งแต่ละองค์ประกอบจะไม่สามารถรับรู้ถึงองค์ประกอบของเลยอร์ก่อนหน้าได้ เว้นแต่องค์ประกอบของเลยอร์ก่อนหน้าที่กำลังโต้ตอบอยู่ นอกจากนี้ ถ้ามีตัวกลาง (intermediate) ระหว่าง Client กับ Server อาทิ เช่น proxy หรือ load balancer และ Client จะไม่ทราบว่ากำลังสื่อสารกับตัวกลางหรือเซิร์ฟเวอร์จริง ๆ ทำให้ไม่ส่งผลกระทบต่อการสื่อสาร และไม่จำเป็นต้องอัปเดต Address ของ Server หรือ Client เรื่อย ๆ อีกทั้งยังช่วยเพิ่มความปลอดภัย



# มาตรฐานของ RESTful API

Uniform interface: เป็นการระบุถึงวิธีการสื่อสารกับ Server โดยไม่คำนึงถึงชนิดของอุปกรณ์ หรือประเภทของแอพพลิเคชัน ซึ่งประกอบด้วย 4 ข้อกำหนด ได้แก่

- Resource-Based: ทุกทรัพยากรจะมี URI เฉพาะเจาะจง ทำให้ต้องร้องขอทรัพยากรผ่าน URI ตัวอย่างเช่น

```
1) https://api.example.com/customers  
2) https://api.example.com/customers/932612...
```

- Manipulation of Resources Through Representations: การดำเนินการกับทรัพยากรผ่านตัวแทน (ไม่อนุญาตให้คิวรีกับฐานข้อมูลโดยตรง) และส่งข้อมูลให้ผู้ใช้ผ่านรูปแบบมาตรฐาน อาทิ เช่น JSON (Passing by Value)

```
{  
  "firstName": "John", "lastName": "Doe"  
}
```

และ Client สามารถปรับแต่งข้อมูลที่ได้รับในรูปแบบมาตรฐาน ก่อนส่งกลับไป Server เพื่ออัพเดตข้อมูลในฐานข้อมูล ส่งผลให้เกิดการ Decouple



# มาตรฐานของ RESTful API

**Uniform interface:** เป็นการระบุถึงวิธีการสื่อสารกับ Server โดยไม่คำนึงถึงชนิดของอุปกรณ์ หรือประเภทของแอพพลิเคชัน ซึ่งประกอบด้วย 4 ชนิด ได้แก่

- **Self-descriptive Messages** (ข้อความอธิบายตนเอง) และ Message มีข้อมูลเพียงพอที่บ่งบอกวิธีการ Process Message ดังกล่าว เพื่อให้ Server สามารถ Process ได้สะดวกและง่าย ตัวอย่างเช่น (1) application/json or application/xml หรือ (2) ข้อตกลง

TASK	METHOD	PATH
Create a new customer	POST	/customers
Delete an existing customer	DELETE	/customers/{id}
Get a specific customer	GET	/customers/{id}
Search for customers	GET	/customers
Update an existing customer	PUT	/customers/{id}

ข้อดีของตัวอย่างที่ (2) เกี่ยวกับโปรโตคอล HTTP คือ ค่อนข้างเป็นที่รู้จักดี ถึงแม้ว่า Client จะไม่ทราบว่ารายละเอียดของระบบ ก็สามารถคาดเดาได้อย่างรวดเร็วว่า Service สามารถทำอะไร โดยสังเกตุจากตัวดำเนินการของ HTTP และ URI



# มาตรฐานของ RESTful API

**Uniform interface:** เป็นการระบุถึงวิธีการสื่อสารกับ Server โดยไม่คำนึงถึงชนิดของอุปกรณ์ หรือประเภทของแอพพลิเคชัน ซึ่งประกอบด้วย 4 ชนิด ได้แก่

- **Hypermedia as the Engine of Application State (HATEOAS):** hypermedia คือ ตัวขับเคลื่อนเพื่อใช้เปลี่ยน state ของ application เปรียบเทียบเหมือนเราท่องเว็บไซต์โดยการ Click ผ่าน Hyperlink เพื่อเปลี่ยน Web Page (= application state) ดังนั้น hypermedia เป็นตัวเชื่อมต่อไปยังส่วนต่าง ๆ (application state อื่น ๆ ) ขณะที่ในมุมมองของ Service เป็น **การเอา output ของ Service 1 ไปเป็น Input ให้อีก Service แทน** ซึ่ง Service ควรทำงานเหมือนเว็บไซต์ที่รับข้อมูลเพียง URI จากนั้นไปตามลิงก์ที่กำหนดไว้ ตัวอย่างเช่น Customer Representation อาจมีส่วน URI เชื่อมไปยัง Order Customer ตัวอย่างข้อมูลที่นำส่ง ได้แก่ Clients จัดส่ง contents, query-string parameters, request headers และ the requested URI ผ่าน Body ของ Request Message ขณะที่ Server จะตอบกลับข้อมูลต่าง ๆ อาทิเช่น response codes และ response headers ผ่าน Body ของ Response Message นอกจากนี้ Service ยังจัดส่ง Link ผ่าน HTTP Header



# มาตรฐานของ RESTful API

**Code on demand** (optional): Server สามารถขยายการทำงานของ Client ขณะ Runtime โดยส่งโค้ดไปยังเซิร์ฟเวอร์ที่ควรรัน (เช่น Java Applets หรือ JavaScript) โดยทั่วไป Web Browser ทำหน้าที่เหมือน Client ของ REST และ Server ส่งผ่าน HTML ให้ Web Browser แสดงผล ซึ่ง Server จะอาศัย Server-side language ที่โปรแกรมขึ้นเพื่อประมวลผลงาน Business Logic ที่ฝัง Server ก่อนการ Render และส่งเป็นข้อมูล HTML ให้ฝัง Client แสดงผลผ่าน Web Browser แต่ถ้าต้องการเพิ่ม Logic บางอย่างที่ทำงานใน Web Browser (ขณะ Runtime) แล้ว Server จะต้องส่งโค้ด JavaScript บางส่วนไปยังฝั่ง Client เพื่อให้ Web Browser ฝั่ง Client ประมวลผลและแสดงผล JavaScript นั้น แต่อย่างไรก็ตาม Code on demand จะก่อให้เกิดการเสีย Visibility ในส่วนของข้อกำหนดของสถาปัตยกรรม REST ดังนั้น Code on demand จึงไม่ใช้ มาตรฐานของ RESTful API ที่บังคับ

```
<!DOCTYPE html> <html>
  <head>
    <script>
      function disableElement() {document.getElementById("btn01").disabled = true;}
    </script>
  </head>
  <body>
    <form> <input type="button" id="btn01" value="OK"> </form>
    <p>Click the "Disable" button to disable the "OK" button:</p>
    <button onclick="disableElement()">Disable</button>
  </body>
</html>
```



# ความแตกต่างระหว่าง REST และ RESTful API

- REST คือ สถาปัตยกรรม
- RESTful เป็นการใช้หรืออ้างถึงสถาปัตยกรรม

การที่ Service ใด ๆ จะเรียกว่าเป็น RESTful API ได้ Service ต้องได้รับการออกแบบให้สอดคล้องกับมาตรฐานของ RESTful API จำนวน 5 ข้อ ยกเว้น ข้อ Code on Demand ที่ไม่บังคับ (Optional)

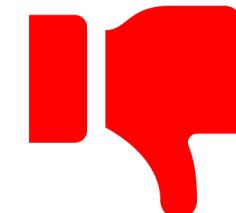


# ข้อดีและข้อควรระวังของ REST



## ข้อดี REST

- โครงสร้างระบบเรียบง่าย
- ง่ายต่อการทดสอบผ่าน Web Browser และรองรับการทำงานร่วมกับ Firewall เนื่องจากทำงานบนโปรโตคอล HTTP
- รองรับการสื่อสารแบบ Request - Response
- ไม่อาศัยตัวกลางเพื่อ coy จัดการต่าง ๆ เพื่อลดความซับซ้อนของสถาปัตยกรรมลง



## ข้อบกพร่องของ REST

- รองรับการสื่อสารแบบ Request – Response เท่านั้น
- ความพร้อมใช้งานของระบบ (Availability) ลดลง เนื่องจากเป็นการสื่อสารโดยตรงระหว่าง Client กับ Server ถ้า Service ล้ม ระบบอาจจะล้มได้เลย
- ต้องทราบ URL ที่ชัดเจนของผู้ให้บริการ หรืออาศัย Service Discovery ในการหาว่าผู้ให้บริการ
- การดึง Resource จากหลาย ๆ ที่ในการร้องขอครั้งเดียวคือความท้าทายในการออกแบบ
- บางครั้งมันยากที่จะหาว่าการทำงานที่หลากหลายรูปแบบที่มีจะไปใช้ HTTP Method ไหนดี



# Outline

- Communication in a monolithic architecture
- Dependency Injection
- Interaction Styles
  - Synchronous vs Asynchronous
  - Request – Response
- Message Format
- Communication Interface
  - Representational State Transfer (REST)”
- Introduction to Spring Framework and Spring Boot



# Spring Framework



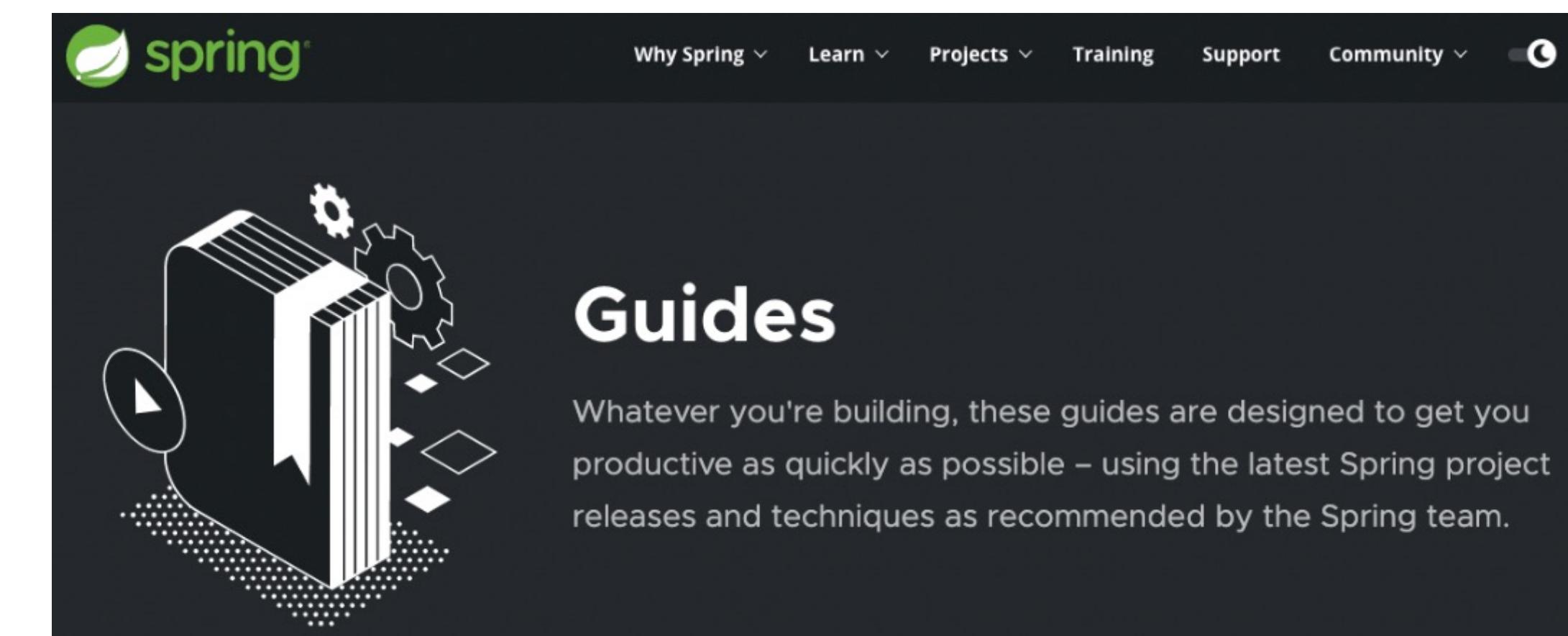
คือ Framework แบบ Open Source สำหรับพัฒนาแอพพลิเคชันด้วยภาษา Java ที่มีน้ำหนักเล็ก (Lightweight) ซึ่งนักศึกษาสามารถใช้งาน Spring Framework เพื่อช่วยพัฒนาระบบในระดับ Enterprise ด้วยภาษา Java ได้ ซึ่ง Spring Framework ได้รวมชุดคำสั่งต่าง ๆ เพื่อรองรับการทำงานในการพัฒนาระบบให้กับนักพัฒนาโปรแกรมไว้มากมาย สิ่งเหล่านี้ทำให้นักพัฒนาสามารถโฟกัสเพียงการพัฒนาในส่วนของ **Business Logic** เพียงเท่านั้น นอกจากนี้ Spring Framework จะช่วยจัดการเรื่องพื้นฐานทั่วไปให้กับนักพัฒนา นอกจากนี้ Spring Framework ยังจัดการโครงสร้างพื้นฐานต่าง ๆ ที่จำเป็นต่อการพัฒนาระบบไว้ให้แล้ว ได้แก่ DB connectivity เป็นต้น อย่างไรก็ตาม นักศึกษาต้องจัดการและกำหนดค่า Config ต่าง ๆ เพื่อให้ Spring Framework สามารถทำงานได้ตามความต้องการของนักศึกษา



# Spring Boot



คือ Opinionated Framework ที่ค่อยอ่านวิความสะดวกแก่นักพัฒนาสำหรับ configuration ค่าต่าง ๆ (ลดลง) ของ Spring Framework นักศึกษาสามารถเรียกใช้ Spring Application ผ่าน Spring Boot ได้อย่างรวดเร็ว ตั้งแต่ ค.ศ. 2021 เป็นต้นมา Spring Boot ได้รับความนิยมสำหรับการใช้เพื่อช่วยพัฒนาระบบแบบ Spring Application นอกจากนี้ ยังอ่านวิความสะดวกในการเพิ่มลด Starter Library ทำให้เรียกใช้งานได้สะดวก และมี Built-in Web Server ทำให้สร้างแอปพลิเคชันแบบแยกส่วนได้ง่ายขึ้น



นักศึกษาสามารถดูตัวอย่างเพิ่มเติมได้จาก <https://spring.io/guides>



# Spring Boot Application Starters

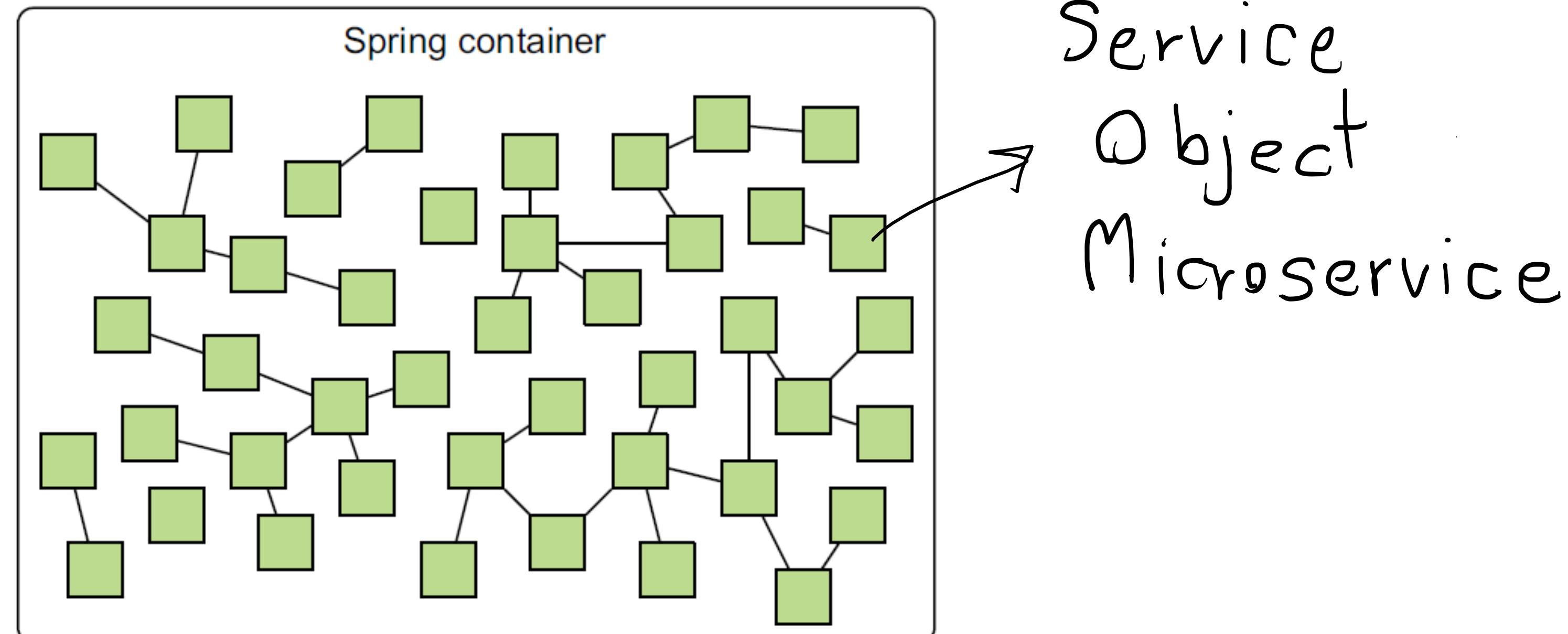
Spring Boot มีชุดคำสั่งเริ่มต้น (Starter) ที่พร้อมใช้งานให้เลือกใช้มากมาย ซึ่งรวมคำสั่งที่เกี่ยวข้องและจำเป็นให้อยู่ในรูปแบบ \*.jar อาทิเช่น

Dependency	Description
spring-boot-starter	สำหรับพัฒนาแอพพลิเคชันด้วย Spring Boot
spring-boot-starter-web	สำหรับพัฒนาเว็บแอพพลิเคชัน
spring-boot-starter-jdbc	สำหรับพัฒนาส่วนติดต่อกับฐานข้อมูลผ่าน JDBC
spring-boot-starter-jpa	สำหรับพัฒนาส่วนติดต่อกับ Java Persistence API (JPA)
spring-boot-starter-data-mongo	สำหรับพัฒนาส่วนติดต่อกับฐานข้อมูล Mongo
spring-boot-starter-data-rest	สำหรับพัฒนา Web API (RESTful API)
spring-boot-starter-ws	สำหรับพัฒนาส่วน Web Service

นักศึกษาสามารถดูเพิ่มเติมได้จาก <https://www.javatpoint.com/spring-boot-starters>



# Spring Container

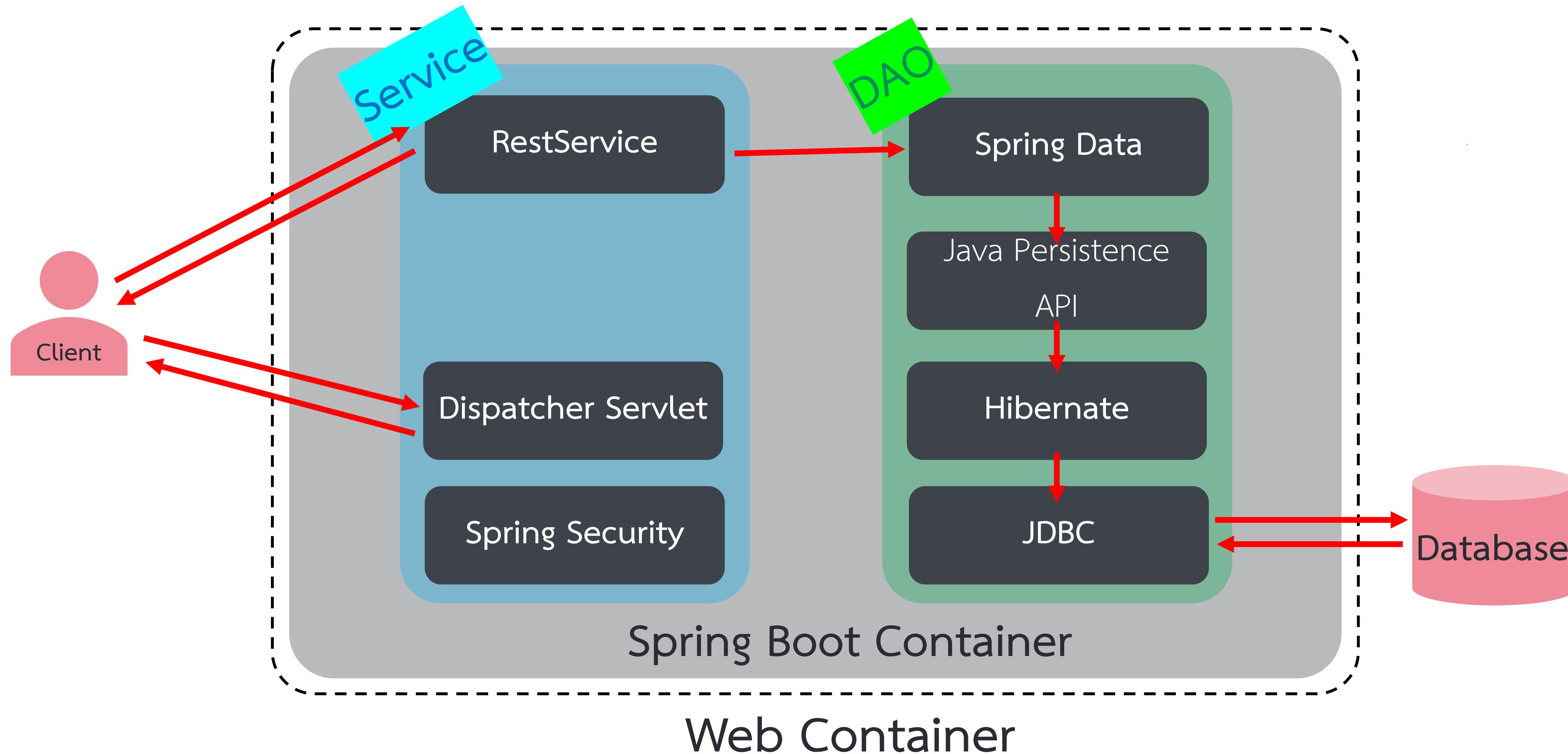


อ้างอิง <https://medium.com/lifeinhurry/what-is-spring-container-spring-core-9f6755966fe9>

ใน Spring Container เป็นที่อยู่อาศัยและจัดการ Object ตั้งแต่ถูกสร้างจนถึงทำลาย โดยนักศึกษาสามารถ (1) แสดงการเชื่อมโยงระหว่าง Object ได้ และ (2) การจำลองการทำงานร่วมกันของ Object เพื่อคุณลักษณะของ Business Logic



# Spring Boot Architecture





# แนะนำ Spring Boot Annotations เปื้องต้น

Spring Boot Annotation คือ Metadata รูปแบบหนึ่งที่ให้ข้อมูลและคำอธิบายเพิ่มเติมเกี่ยวกับโปรแกรม ไม่ว่าจะเป็นส่วนหนึ่งของโปรแกรมที่เรากำลังพัฒนา ไม่ส่งผล และไม่เปลี่ยนแปลงการทำงานของโค้ด โดยที่ Annotation ต่าง ๆ จะอยู่ใน Package

```
import org.springframework.boot.SpringApplication;
```



# แนะนำ Spring Boot Annotations เปื้องต้น

**@SpringBootApplication** คือ Annotation ที่บ่งบอกว่าคลาสนั้นเป็น Application class และมีเมธอด main() อู่

```
@SpringBootApplication  
  
public class MyApp{  
    public static void main(String[] args) {  
        SpringApplication.run(MyApp.class, args);  
    }  
}
```

เป็นการรวมหน้าที่ 3 อย่างของ annotation ต่อไปนี้เข้าด้วย ได้แก่

- **@ComponentScan** คือ การค้นหา (1) Spring component และ (2) Configuration class ใน package ของ Application class และ package อื่นที่เกี่ยวข้อง
- **@EnableAutoConfiguration** คือ การอนุญาตให้ Spring Boot จัดการ configure ต่าง ๆ ใน JAR ไฟล์ ของ classpath เองโดยอัตโนมัติ
- **@Configuration** คือ การอนุญาตกำหนด configure และ definition ใน Java ไฟล์ เนื่องจากใน Spring เวอร์ชัน 3 เป็นต้นมาอนุญาตให้เราสามารถกำหนด configure และ definition ของแต่ละ Object ภายนอก \*.xml ได้



# Skeleton Project

pom.xml คือไฟล์สำหรับกำหนด Config ต่าง ๆ ของ Maven ในโปรเจค

```
❶ pom.xml
 1  <?xml version="1.0" encoding="UTF-8"?>
 2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
 3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0
 4  <modelVersion>4.0.0</modelVersion>
 5  <parent>
 6      <groupId>org.springframework.boot</groupId>
 7      <artifactId>spring-boot-starter-parent</artifactId>
 8      <version>2.5.0</version>
 9      <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.example.demo1</groupId>
12  <artifactId>demo</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>demo</name>
15  <description>Demo project for Spring Boot</description>
16  <properties>
17      <java.version>11</java.version>
18  </properties>
19  <dependencies>
20      <dependency>
21          <groupId>org.springframework.boot</groupId>
22          <artifactId>spring-boot-starter-web</artifactId>
23      </dependency>
24
25      <dependency>
26          <groupId>org.springframework.boot</groupId>
27          <artifactId>spring-boot-starter-test</artifactId>
28          <scope>test</scope>
29      </dependency>
30  </dependencies>
31
32  <build>
33      <plugins>
34          <plugin>
35              <groupId>org.springframework.boot</groupId>
36              <artifactId>spring-boot-maven-plugin</artifactId>
37          </plugin>
38      </plugins>
39  </build>
40
41 </project>
```

บอกรายละเอียดของโปรเจค รุ่น JAVA และ รุ่นของ Spring Boot

บอกรายละเอียดคำสั่งของ Spring Boot Starter Web Dependency

บอกรายละเอียดคำสั่งของ Spring Boot Starter Test Dependency

บอกรายละเอียดคำสั่งของ Spring Boot Specific Maven Plugins for building and deploying



# Spring Bootstrap Class

คือ คลาสที่ Spring Boot จะเรียกใช้งานเพื่อกำหนดค่าเริ่มต้นต่าง ๆ และสั่งให้ Service เริ่มทำงาน โดยที่ Spring Boot จะใช้ Annotation เพื่อทำให้การตั้งค่าต่าง ๆ ง่ายขึ้น

```
src > main > java > com > example > demo1 > demo > DemoApplication.java > ...
1 package com.example.demo1.demo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class DemoApplication {
8     Run | Debug
9     public static void main(String[] args) {
10         SpringApplication.run(DemoApplication.class, args);
11     }
12 }
13
14 }
```

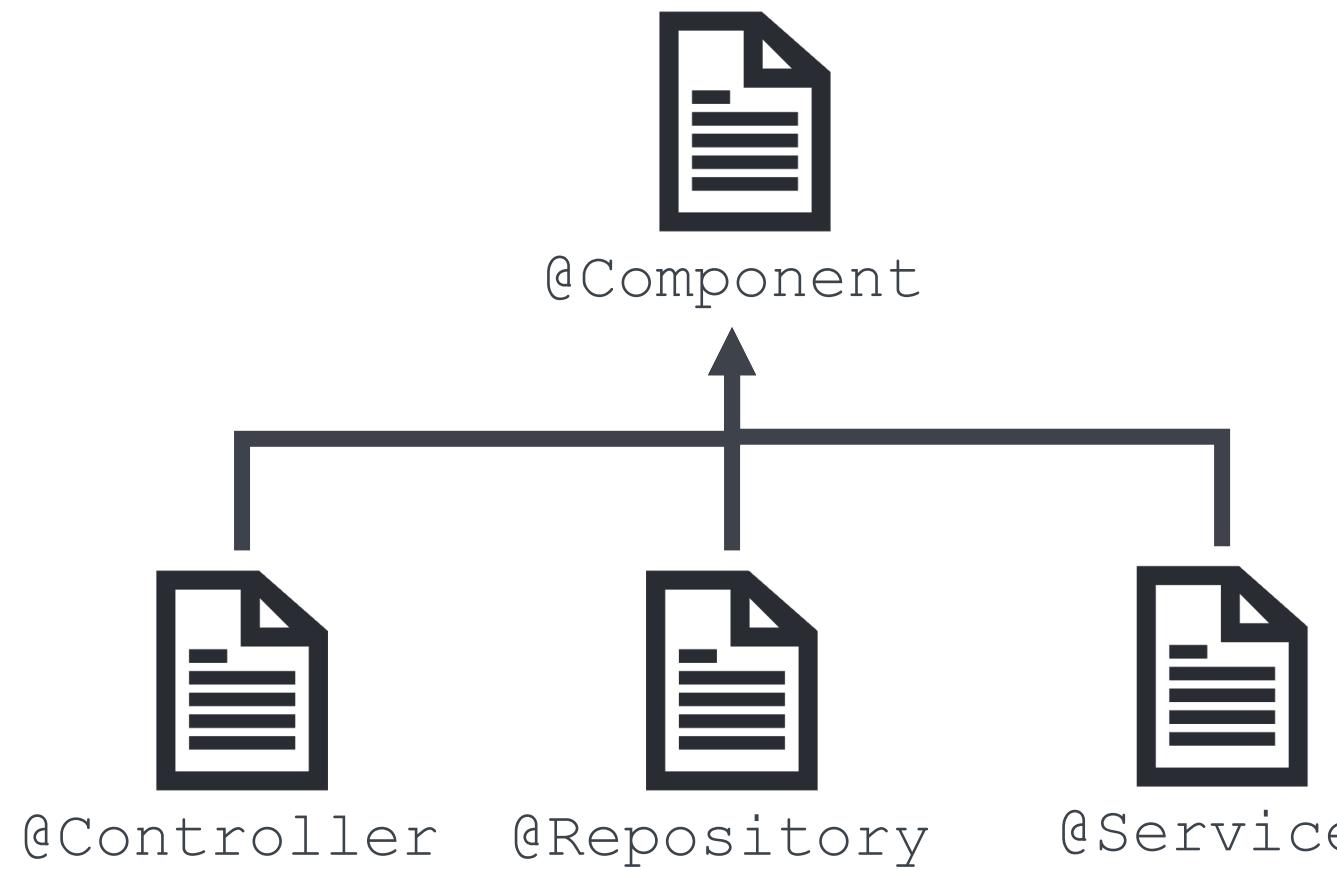
@SpringBootApplication คือ Annotation ที่ใช้บ่งบอก Spring Framework ว่าคลาสนี้เป็น Spring Bootstrap Class ของโปรเจค ซึ่งถือว่าเป็น Core Initialization logic สำหรับ Service นอกจากนี้ยังเป็นตัวบ่งบอกว่าเมธอด main() ของ Service นี้อยู่ที่นี่

ใช้รับค่าผ่าน Command line

เรียกให้ Spring Container เริ่มทำงานและ return ตัว Spring ApplicationContext object



# Spring Bean



@Autowired เป็น Annotation ที่ใช้บ่งบอก Spring Framework ว่าให้หา Object ที่เกี่ยวข้องหรือสร้างใหม่มากำหนดให้ ซึ่งหมายความว่าเราสามารถใช้งานได้ ถึงแม้ว่าเราจะยังไม่ได้สร้าง Object ดังกล่าว ซึ่งการทำงานแบบจะเรียกว่า “Inject object” หรือ “Dependency Injection”

คลาส Java ใน Spring Framework ที่รองรับคุณสมบัติ Dependency Injection (หมายความว่า “คลาสดังกล่าวสามารถ Inject Object มาใช้งานได้ เพียงประกาศ object ดังกล่าวไว้ก่อนเท่านั้น”) จะถูกเรียกว่า "Spring Bean" ประกอบด้วย 4 ชนิด ได้แก่ (1) ทั่วไป (2) ส่วนติดต่อผู้ใช้งาน (3) จัดการข้อมูลและ (4) แนวคิดทางธุรกิจ เราสามารถกำหนดได้ด้วย 4 Annotation ระดับคลาสดังนี้

Annotation	คุณสมบัติ
@Component	การระบุว่า Object ของคลาสนี้เป็น Spring Bean แบบทั่วไป
@Controller	การระบุว่า Object ของคลาสนี้เป็น Spring Bean ที่มีหน้าที่จัดการส่วน Presentational Layer หรือ View
@Repository	การระบุว่า Object ของคลาสนี้เป็น Spring Bean ที่มีหน้าที่จัดการส่วน Persistence Layer หรือ กίng ๆ Model
@Service	การระบุว่า Object ของคลาสนี้เป็น Spring Bean ที่มีหน้าที่จัดการส่วน Business Layer หรือ Controller



# Spring Controller Class

คือ คลาสที่เชื่อมโยงระหว่าง (1) การร้องขอรับบริการผ่านโปรโตคอล HTTP กับ (2) จำแมรอดที่เรียกใช้งานเพื่อประมวลผลบริการนั้น

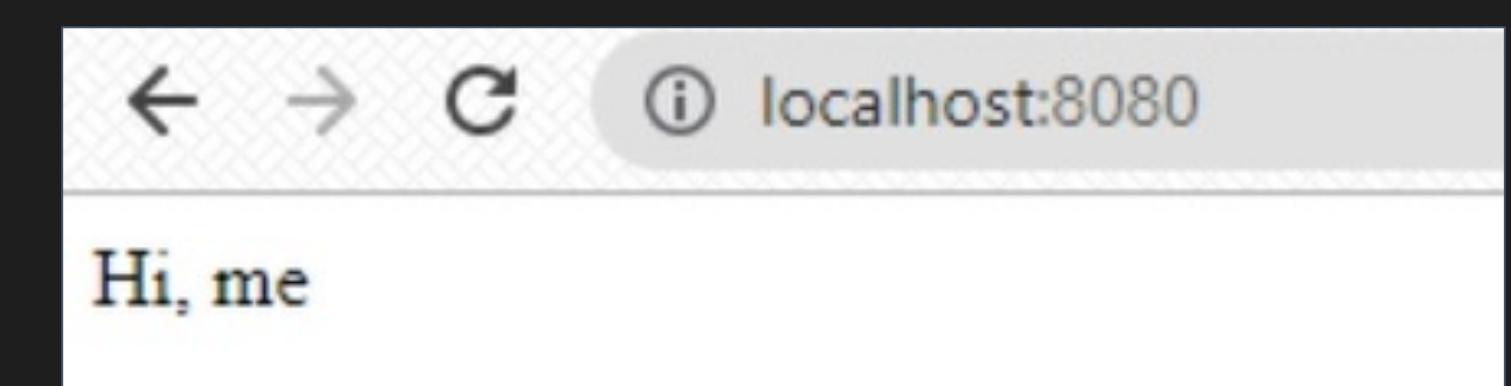
src > main > java > com > example > demo1 > demo > App01.java > App01 > sayHi()

```
1 package com.example.demo1.demo;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4 import org.springframework.web.bind.annotation.RestController;
5 @CrossOrigin(origins = "*", allowedHeaders = "*")
6 @RestController
7 public class App01 {
8
9     @RequestMapping("/")
10    public String sayHi(){
11        return "Hi, me";
12    }
13    @RequestMapping("/HiYou")
14    public String sayHiYou(){
15        return "Hi, you";
16    }
17 }
```

เพื่อนำมาติดต่อเรียกใช้ Service ได้จากทุก domain

ระบุว่าให้คลาสนี้เป็น Controller ของ Service นี้

กำหนด URL ของ HTTP Request สำหรับเรียกใช้งาน ผ่านตัวดำเนินการ GET





# Spring REST Annotation

**@RestController** คือ Annotation ที่ผสมการทำงานระหว่าง @Controller และ @ResponseBody เข้าด้วยกัน

- @Controller เป็น Annotation ระดับคลาส (ใช้กับที่คลาส) ที่บ่งบอกว่าคลาสหรือ component ดังกล่าว เป็นตัวจัดการ Web Request ที่เข้ามา
- @ResponseBody เป็น Annotation ที่ใช้บ่งบอก Spring Boot ให้ (1) เชื่อมโยงค่าส่งคืนกับเมธอดไปยังส่วน Body ของ Response Message และ (2) จัดการ serialize ข้อมูลหรือ object ที่จะคืนค่าให้อยู่ในรูปแบบ JSON หรือ XML ก่อนนำส่ง
- @Service เป็น Annotation ที่ใช้บ่งบอกว่าคลาสหรือ component ดังกล่าว เป็นตัวเก็บ business logic และถูกใช้ใน service layer

```
import org.springframework.web.bind.annotation.*;  
  
@RestController  
public class DemoController {  
  
}
```



# Spring REST Annotation

@RequestMapping คือ Annotation ระดับเมธอด ที่เชื่อมโยงระหว่าง Web Request เข้ากับเมธอด ซึ่งกระบวนการนี้จะเรียกว่า “routing”

```
@RestController  
public class CarController {  
  
    @RequestMapping(value = "/cars", method = RequestMethod.GET)  
    public List<Car> getCars() {  
        return cars;  
    }  
}
```

ประกอบไปด้วย 2 พารามิเตอร์ ได้แก่

- value เป็นส่วนที่ใช้กำหนด URL ในการเรียกใช้งานเมธอด getCars()
- method เป็นส่วนที่ใช้กำหนดตัวดำเนินการ หรือรูปแบบการสื่อสาร ได้แก่ RequestMethod.GET, RequestMethod.POST, RequestMethod.PUT และ RequestMethod.DELETE เป็นต้น โดยที่ Spring Boot จะกำหนดเป็น GET ถ้าไม่มีการระบุ



# Spring REST Annotation

ตัวอย่างการกำหนด method ของ @RequestMapping ให้รองรับการสื่อสารในรูปแบบ GET และ POST โดยอาศัยเครื่องหมาย {...} มาช่วย

```
@RestController  
public class CarController {  
    @RequestMapping(value = "/cars", method = {RequestMethod.GET, RequestMethod.POST})  
    public List<Car> getCars() {  
        return cars;  
    }  
}
```

นอกจากนี้ Spring Boot จะกำหนด method ของ @RequestMapping ให้การสื่อสารเป็นรูปแบบ GET เมื่อผู้พัฒนาไม่ได้กำหนด

```
@RestController  
public class CarController {  
    @RequestMapping(value = "/cars")  
    public List<Car> getCars() {  
        return cars;  
    }  
}
```



# Spring REST Annotation

นอกจากนี้ path เป็นอีกหนึ่งตัวเลือกที่ทำงานเหมือนกับ value แต่สามารถรองรับการกำหนด URL แบบ Ant-style URL mappings โดยมีข้อกำหนดดังนี้

สัญลักษณ์	ความหมาย
?	ตรงเพียงหนึ่งตัวอักษร
*	ตรงตั้งแต่ศูนย์ตัวอักษรขึ้นไป
**	ตรงตั้งแต่ศูนย์ directory ขึ้นไป
{spring: [a-z] +}	อาศัยหลักการ regexp

ตัวอย่างเช่น

ตัวอย่างคำสั่ง	อธิบาย
com/t?st.jsp	com/test.jsp หรือ com/tast.jsp หรือ com/txst.jsp
com/**/test.jsp	ได้ test.jsp ทุกไฟล์ภายใต้ root folder ของ com
org/test/**/*.jsp	ได้ .jsp ทุกไฟล์ภายใต้ root folder org/test
org/**/servlet/bla.jsp	org/aa/servlet/bla.jsp หรือ org/aa/bb/servlet/bla.jsp หรือ org/servlet/bla.jsp



# Spring REST Annotation

```
@RestController  
public class CarController {  
    @RequestMapping(path = "{name:[a-z]+.do}", method = RequestMethod.GET)  
    public String test(@PathVariable("name") String name) {  
        return name;  
    }  
}
```

The screenshot shows the Postman application interface. At the top, there is a header bar with 'GET' selected as the method, the URL 'http://localhost:8080/tara.do', and a 'Send' button. Below the header, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', 'Settings', and 'Cookies'. The 'Params' tab is active, showing a table with one row containing 'Key' and 'Value' columns. Under the 'Headers' tab, there are six entries listed. In the 'Body' tab, there is a note: 'Body is empty because you selected "Text" as the type in the "Content-Type" header.' At the bottom, the response status is shown as '200 OK' with a response time of '232 ms' and a size of '214 B', along with a 'Save Response' button. The bottom navigation bar includes 'Pretty', 'Raw', 'Preview', 'Visualize', 'Text', and a search icon.



# Spring REST Annotation

```
@RestController  
public class CarController {  
    @RequestMapping(path = "{name:[a-z]+}.do", method = RequestMethod.GET)  
    public String test(@PathVariable("name") String name) {  
        return name;  
    } }
```

The screenshot shows the Postman interface with the following details:

- Method: GET
- URL: <http://localhost:8080/tara.do>
- Params tab selected
- Query Params table:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		
- Headers tab (5 headers listed)
- Body tab selected, showing the response body: `1 tara`
- Response status: 200 OK, 173 ms, 167 B



# Spring REST Annotation

นอกจากนี้ นักศึกษาสามารถใช้ Annotation ระดับเมธอดที่เชื่อมโยงระหว่าง Web Request เข้ากับเมธอดตัวอื่น ๆ ที่มีการระบุ method ที่แน่นอนแทน @RequestMapping ได้ดังนี้

```
@GetMapping  
@PostMapping  
@PutMapping  
@DeleteMapping  
@PatchMapping
```

นักศึกษาสามารถหารายละเอียดเพิ่มเติมได้จาก <https://zetcode.com/all/#springboot> และ <https://www.javatpoint.com/spring-boot-annotations>



# Spring REST Annotation

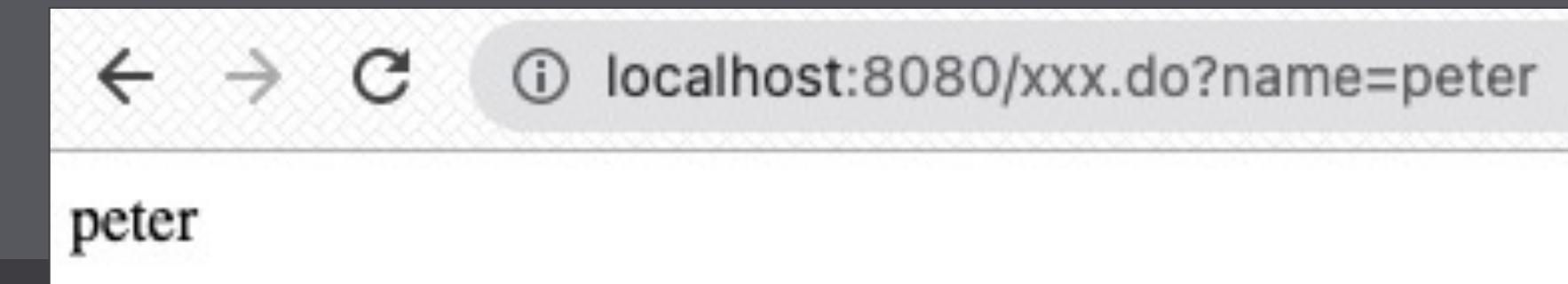
**@PathVariable** และ **@RequestParam** เป็น Annotation ที่ช่วยสกัดค่าพารามิเตอร์ที่ส่งมาผ่าน URL และ/หรือ URI ของ Web Request เข้ามาสู่เมธอด แต่จะแตกต่างกันตรงที่

- **@PathVariable** หมายความกับการทำงานแบบ RESTful API และสกัดค่าพารามิเตอร์ด้วย path variable ด้วย {...}

```
@RequestMapping(value = "/name.do/{name}", method = RequestMethod.GET)
public String test(@PathVariable("name") String name) {
    return name;
}
```

- **@RequestParam** หมายความกับการทำงานแบบ Web Application และสกัดค่าพารามิเตอร์ด้วย query parameter

```
@RequestMapping(value = "/xxx.do", method = RequestMethod.GET)
public String test(@RequestParam("name") String name) {
    return name;
}
```

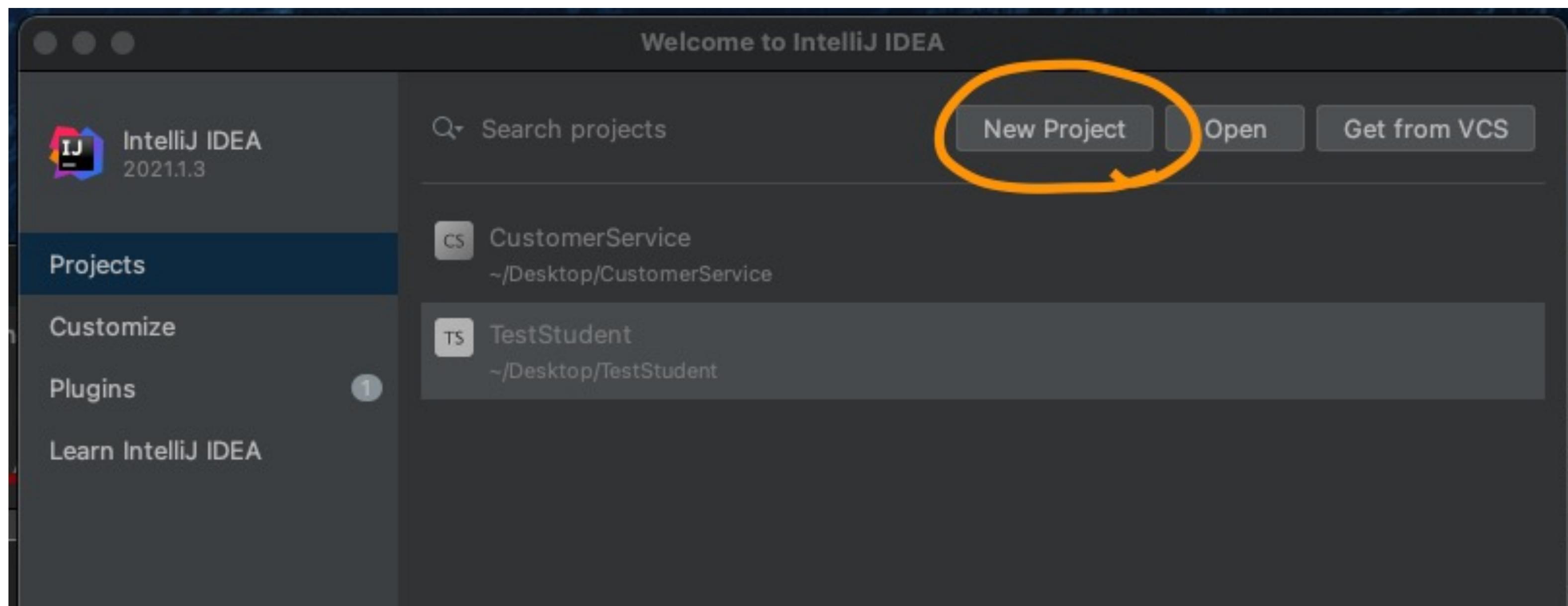




# การสร้างโปรเจค Spring ด้วย IntelliJ IDEA

กำหนดให้นักศึกษาสร้าง Service ด้วย Spring ผ่านโปรแกรม IntelliJ IDEA และเรียกใช้งานผ่าน Web Browser

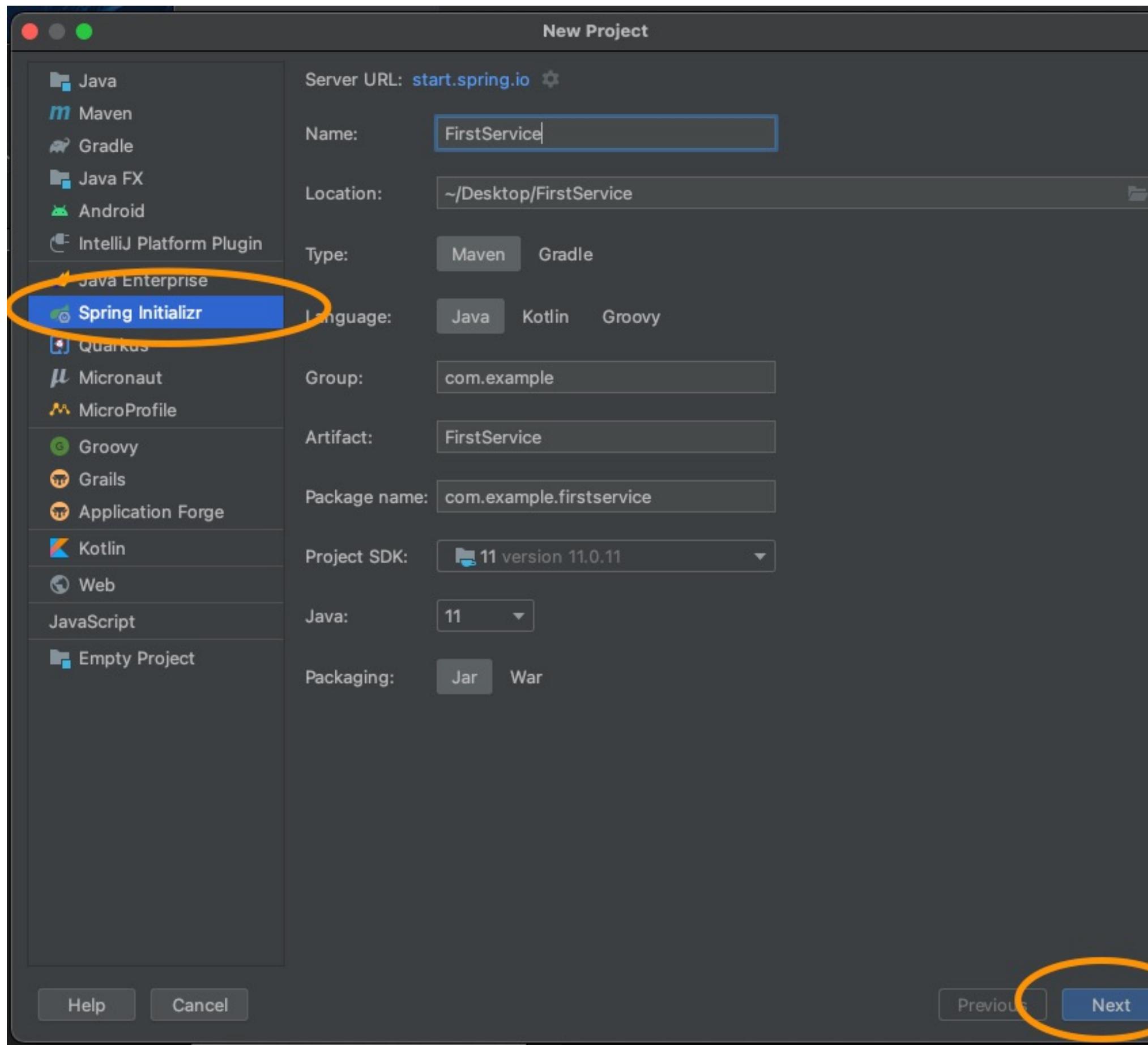
ขั้นที่ 1 ให้นักศึกษาเปิดโปรแกรม IntelliJ IDEA จากนั้นเลือก New Project ดังภาพ





# การสร้างโปรเจค Spring ด้วย IntelliJ IDEA

ขั้นที่ 2 เลือก Spring Initializer และกรอกข้อมูลให้ครบดังคำอธิบายในตาราง ก่อนกดปุ่ม Next ดังภาพ

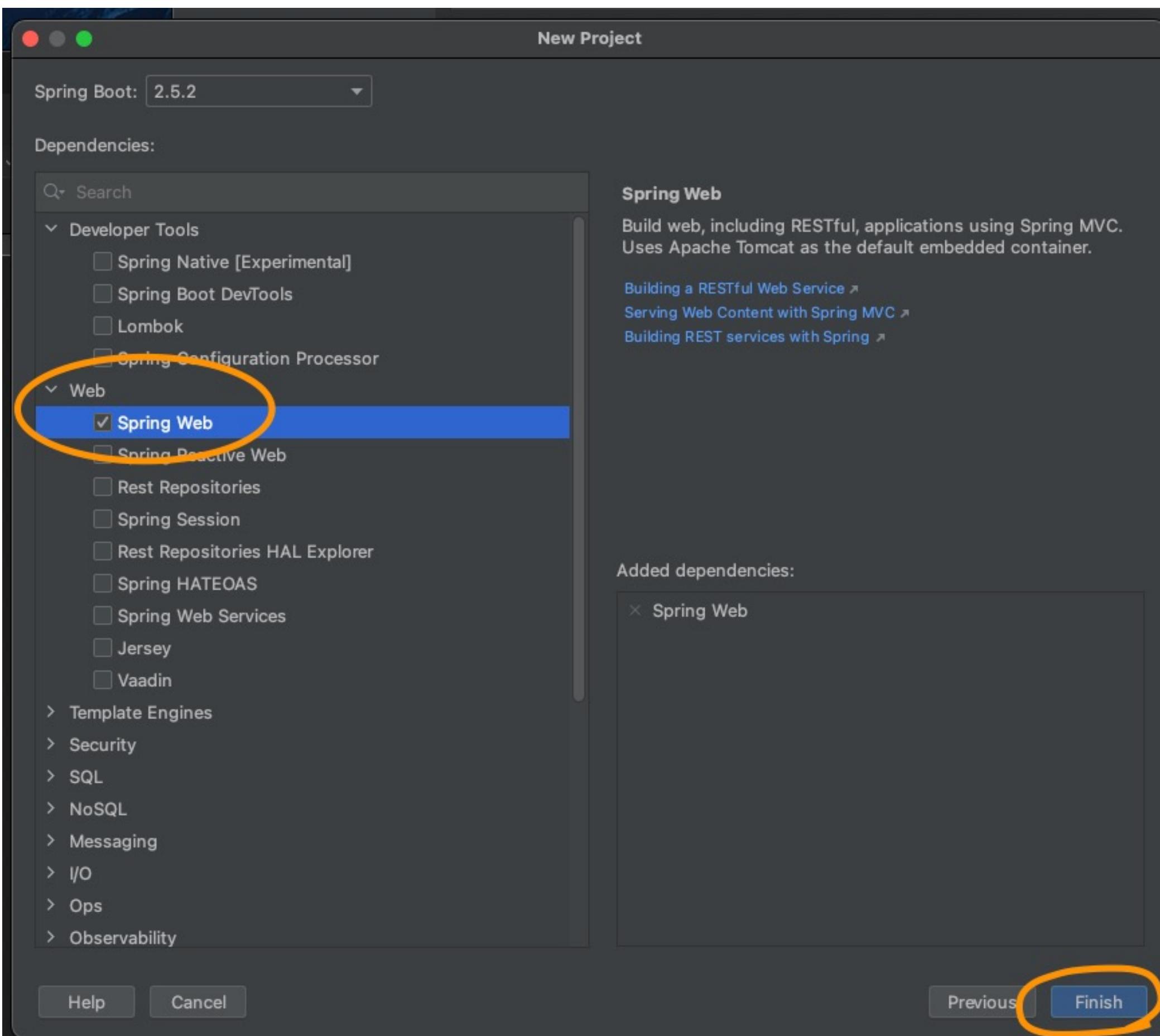


ชื่อ	คำอธิบาย
Name	ระบุชื่อโปรเจค
Location	ระบุสถานที่เก็บไฟล์
Type	ระบุ Framework
Language	ระบุภาษาที่ใช้พัฒนา
Project SDK	ระบุรุ่น JDK
JAVA	ระบุรุ่นภาษา
Packing	ระบุรูปแบบการรวมไฟล์ก่อนการ Deploy



# การสร้างโปรเจค Spring ด้วย IntelliJ IDEA

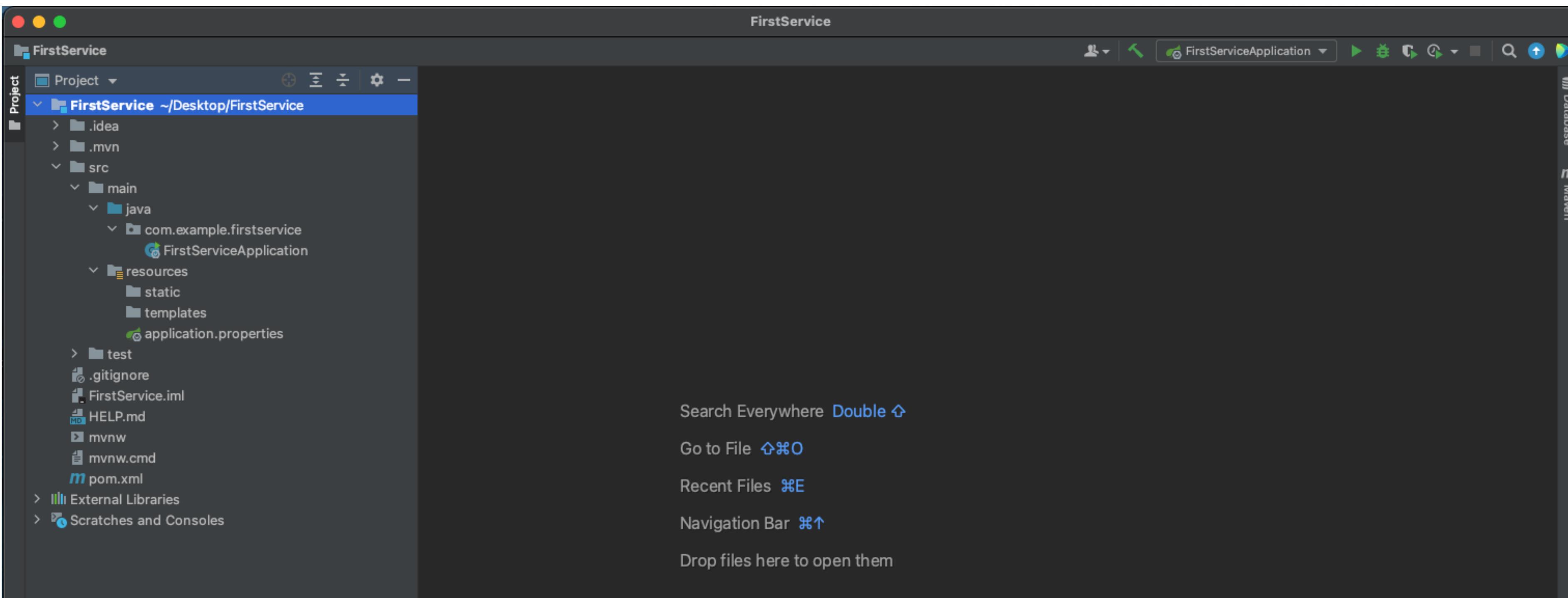
ขั้นที่ 3 คลิก Web และเลือก Spring Web จากนั้น กด Finish





# การสร้างโปรเจค Spring ด้วย IntelliJ IDEA

## ขั้นที่ 4 pragกูหน้าต่างโปรแกรมดังภาพ





# การสร้างโปรเจค Spring ด้วย IntelliJ IDEA

ขั้นที่ 5 ถ้านักศึกษารันเมธอด main() ของไฟล์ที่มี @SpringBootApplication ตัว Spring boot จะไปสร้าง Object และ Inject จากไฟล์ที่มี @RestController ไว้ให้ Object ทำงานโดยอัตโนมัติ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "FirstService". The "FirstServiceApplication.java" file is selected in the editor.
- Editor:** Displays the Java code for the main application class:

```
import ...  
@SpringBootApplication  
public class FirstServiceApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(FirstServiceApplication.class, args);  
        System.out.println("Hello A Bank");  
    }  
}
```
- Run Tab:** Shows the console output of the application's execution:

```
2021-07-19 20:41:14.456  INFO 4744 --- [           main] c.e.f.FirstServiceApplication          : Starting FirstServiceApplication using Java 11.0.11 on Mac-mini-khxng-Tara  
2021-07-19 20:41:14.458  INFO 4744 --- [           main] c.e.f.FirstServiceApplication          : No active profile set, falling back to default profiles: default  
2021-07-19 20:41:15.631  INFO 4744 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)  
2021-07-19 20:41:15.639  INFO 4744 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]  
2021-07-19 20:41:15.639  INFO 4744 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.48]  
2021-07-19 20:41:15.688  INFO 4744 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]      : Initializing Spring embedded WebApplicationContext  
2021-07-19 20:41:15.688  INFO 4744 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1103 ms  
2021-07-19 20:41:16.046  INFO 4744 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''  
2021-07-19 20:41:16.069  INFO 4744 --- [           main] c.e.f.FirstServiceApplication          : Started FirstServiceApplication in 2.078 seconds (JVM running for 2.845)  
Hello A Bank
```



# การสร้างโปรเจค Spring ด้วย IntelliJ IDEA

## สร้าง RESTful API Service พื้นฐาน

ขั้นที่ 1 สร้างไฟล์จavadoc ใหม่ โดยนักศึกษามองว่าไฟล์นี้เป็น Controller ของ Service

ขั้นที่ 2 ใส่ annotation “@RestController” บนชื่อคลาส

ขั้นที่ 3 สร้างเมธอด พร้อมด้วยกำหนดรูปแบบการสื่อสาร (GET/POST/DELETE/PUT) และพารามิเตอร์ของ Service สิ่งนี้สามารถเทียบเคียงได้กับบริการที่ Service ได้เตรียมไว้ให้บริการ

```
@RestController  
public class TestRestAPI {  
    @RequestMapping(value = "/helloWorld", method = RequestMethod.GET)  
    public String helloWorld() {  
        return "Hello World";  
    }  
}
```

จะพบว่า Service ชื่อ TestRestAPI มีบริการจำนวน 1 บริการ คือ helloWorld ที่มีการสื่อสารแบบ GET และไม่มีการรับพารามิเตอร์ใด