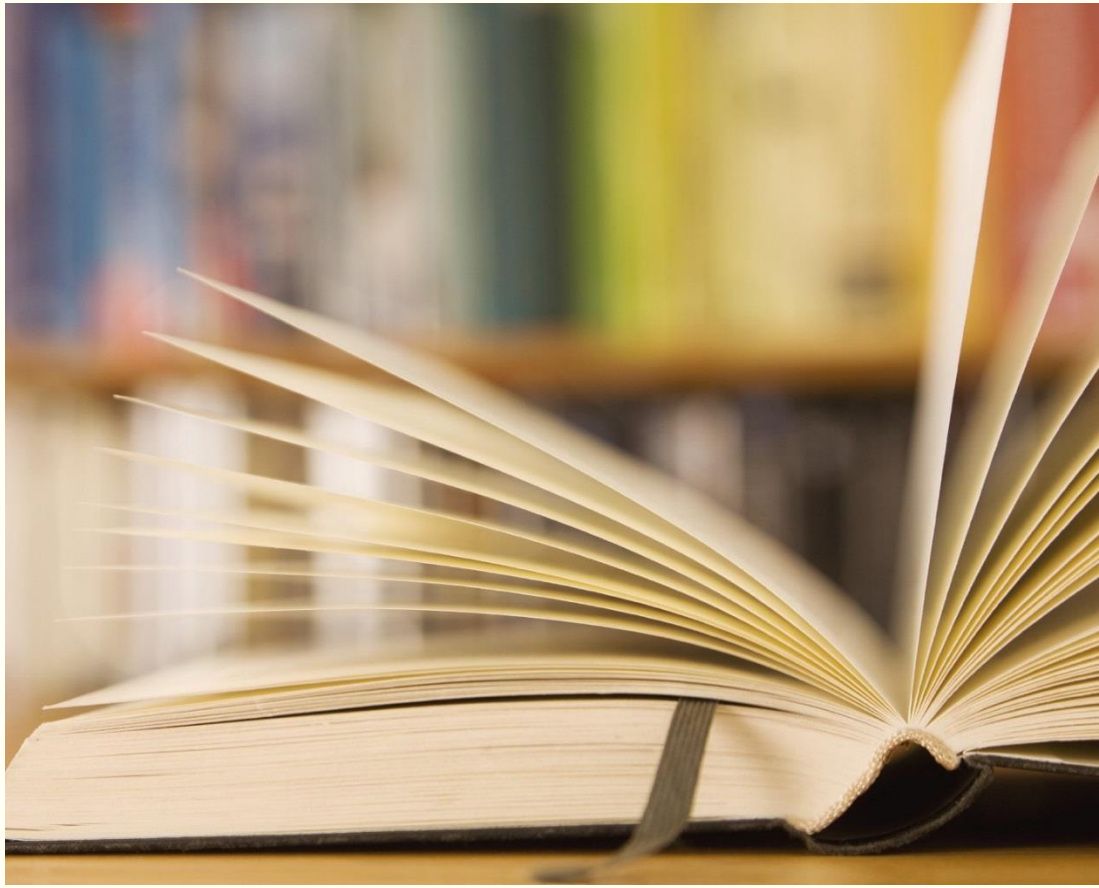
ENTERPRISE ARCHITECTURE

Najeeb Najeeb, PhD



LESSON 10 WEB SERVICES

What is a Web Service

- Service delivered over the web?
- Application A can be used to send SMS
  - Can we reuse this application?
  - UI of my application is different than the UI of SMS App.
  - If SMS app uses layers then can we reuse parts of it?
    - Copy the layer
    - What about layer dependencies
    - What problem can this also have?
- Software system designed to support interoperable machine-to-machine interaction over a network
  - https://www.w3.org/TR/ws-arch/#whatis

Request / Response

- Web Service data exchange
  - Input needs to be sent (request)
  - Output is sent back (response)
- Platform independent request/response
  - This results in platform independent consumer and provider
- Data Exchange Format
  - XML
  - JSON
- How are formats agreed on?
  - What request to send and what to expect?
  - Service definitions
    - Request/Response Format
    - Request Structure
    - Response Structure
    - Endpoint (where is the data available)

Terminology

- Request
  - Input
- Response
  - Output
- Message Exchange Format
  - XML, JSON
- Service Provider / Consumer
- Service Definition
- Transport
  - HTTP
  - MQ

They are Two But Not Really

- Web Services
- SOAP
  - Simple Object Transfer Protocol
  - A restriction on the format of XML exchanged
- REST
  - REpresentational State Transfer
  - An architectural approach

SOAP

SOAP

- Not really Simple Object Access Protocol
- A term for a specific way to create web services
- Defines a request/response structure
  - SOAP envelope
    - SOAP header
    - SOAP body
- No restrictions on transport
  - Can be over http
  - Can be over queues
- Service Definition
  - WSDL (Web Service Definition Language)
  - Endpoints, operations, request structure, response structure

REST

REST

- Roy Fielding
  - Developed HTTP
  - Part of his PhD dissertation
- The Misunderstanding of his work
- This is not the first time
  - Waterfall !!!

REST

- HTTP
  - HEADER
  - BODY
  - METHOD
    - GET
    - POST
    - PUT
    - DELETE
  - STATUS
- Data Format
  - No restriction (JSON is popular)
- Transport
  - HTTP only
- Service Definition
  - No standard (WADL, Swagger, ...)

• REpresentaional State Transfer

- Resource
  - URL Uniform Resource Location
- Representation
- State
- Transfer

Compare

SOAP

- XML Restriction
- Data Exchange: XML
- WSDL
- HTTP or MQ
- Need to define WSDL and XML parsing

REST

- Architectural Style
- No Strict Data Exchange Format
  - JSON is popular
- No standard
  - WADL, Swagger
- HTTP
- Easier to Implement (can get away with nothing)

SPRING BOOT

Spring

- What dependencies are needed (projects to use)
- Select a version for each one
  - Figure out version dependencies
- Implement default exception handling
- Full Spring configuration
- Configure JPA
- Replicate all configurations for all profiles

Spring Boot

IS

- Opinionated Spring framework
- Application split into many micro services
  - Fast development of each service
- Provide common non-functional features

NOT

- No Code Generation
- Not an application server
- Not a web server

Starter Projects

- When working with Spring you may need to include different spring projects (dependencies)
- You will also need to configure each project
- You need to check compatibility between these projects
- Spring Projects are already defined sets of projects, with the proper compatible versions selected and configured with some conventional configurations.
- Embedded Servers
- Built in metrics and health checks (analytics)
- Externalized configuration
  - Support for profiles

Spring Initializer

- https://start.spring.io/
- Auto Configuration
  - Check Classpath
  - Application Configuration (frameworks)
- application.properties
  - logging.level.org.springframework = debug
  - Run in debug mode, get to know matching configurations and non-matching

Starter Projects

- Add Web
- Check spring-boot-starter-web
- Do you see any issues, or concerns you may have?
- Starter-json brings json binding
  - Send a java object get a json object
- All starter projects inherit from spring-boot-starter
- When solving a problem check if there is a starter for it

Which one is Better

Spring

- Testing, DI, loosely coupled application
- Reduce (eliminate) boiler plate code when using frameworks
  - Templating
- Good integration with frameworks

Spring Boot

- Reduce configuration needs
- Example of Convention over Configuration in Spring framework configuration
- Can we autoconfig based on added dependencies
- Starter projects
- Monitoring projects (or starters)

REST

Setup Application

- Dependencies
  - Web (REST Application)
- Start Application
- Tomcat starts
  - Tomcat initialized with port(s): 8080 (http)

REST

- A style of software architecture for distributed systems.
- Currently built on top of HTTP
  - Not the original intent
- Create a Student
  - POST /students
- Delete a Student
  - DELETE /students/{student_id}
- Get all Students
  - GET /students
- Get one Student
  - GET /students/{student_id}

Nested Usage

- Students
- Courses
  - GET all courses
    - GET /students/{student_id}/courses
  - Create a course
    - POST /students/{student_id}/courses
  - Get one course
    - GET /students/{student_id}/courses/{course_id}
- Guidelines not musts but this is the convention

Hello World

Requirements

- Hello World!
- Return message "Hello World!" On
- GET method
- URI /hello

Controller

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

```
@GetMapping(path= "/hello")
```

Student

Student

- Student class
- Constructor
- Getters & setters
- Add property
  - logging.level.org.springframework= debug

Controller

```
@GetMapping(path= "/student")
public Student getStudent() {
	return new Student("Jack", 3.0f);
}
```

Path Variables (request params)

Student

- GET
- /hello/{name}

Controller

```
@GetMapping(path= "/hello/{name}")
public String helloName(@PathVariable
String name) {
        return "Hello "+name;
}
```

Path Variables (request params)

Student

```
@Component
public class StudentManager {
	private List<Student> students;
	public StudentManager() {
		this.students = new ArrayList<>();
		students.add(new Student("Jack", 3.0f));
		students.add(new Student("John", 2.0f));
		students.add(new Student("Jill", 3.5f));
		students.add(new Student("James", 2.9f));
		students.add(new Student("Jasmin", 3.1f));
	}
	public List<Student> getStudents() {
		return students;
	}
	public Student save(Student student) {
		students.add(student);
		return student;
	}
	public Student findOne(int id) {
		Student foundStudent= null;
		for (Student student : students) {
			if (student.getId() == id) {
				foundStudent= student;
				break;
			}
		}
		return foundStudent;
	}
}
```

Controller

```
@RestController
public class StudentResource {
	@Autowired
	StudentManager studentManager;
	@GetMapping("/students")
	public List<Student> getAllStudents() {
		return studentManager.getStudents();
	}
	@GetMapping("/students/{student_id}")
	public Student getOneStudent(@PathVariable int student_id) {
		return studentManager.findOne(student_id);
	}
}
```

Add User

```
@PostMapping(value = "/students")
public void addOneStudent(@RequestBody Student student) {
	Student savedStudent= studentManager.save(student);
	int savedStudentId= savedStudent.getId();
	URI saveStudentUri=
ServletUriComponentsBuilder.fromCurrentRequest().path("/{student_id}").buildAndExp
and(savedStudentId).toUri();
	return ResponseEntity.created(saveStudentUri).build();
}
```

Error Handling

Student 500 is not found

- Request students/500
  - You get 200 ok !!! But this does not exists

```
@GetMapping("/students/{student_id}")
public Student
getOneStudent(@PathVariable int
student_id) {
	Student student=
studentManager.findOne(student_id);
	if (student == null) {
		throw new
UserNotFoundException("id:"+student_id);
	}
	return student;
}
```

Exception Class

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class UserNotFoundException extends
RuntimeException{
	public UserNotFoundException(String
message) {
		super(message);
	}
}
```

Get rid of the trace

```
server.error.include-stacktrace=never
```

Delete Student

StudentManager

```
public Student removeOne(int student_id)
{
	Iterator<Student>  iterator= students.iterator();
	Student deletedStudent= null;
	while (iterator.hasNext()) {
		Student student= iterator.next();
		if (student.getId() == student_id) {
			iterator.remove();
			deletedStudent= student;
		}
	}
	return deletedStudent;
 }
```

StudentResource

```
@DeleteMapping("/students/{student_id}")
public void removeOndStudent(@PathVariable int student_id) {
	Student student= studentManager.removeOne(student_id);
	if (student == null) {
		throw new UserNotFoundException("id:"+student_id);
	}
}
```

INTERNATIONALIZATION I18N

LocaleResolver

SpringRestApplication

```
@SpringBootApplication
public class SpringRestApplication {
	public static void main(String[] args) {
		SpringApplication.run(SpringRestAp
plication.class, args);
	}
	@Bean
	public LocaleResolver localeResolver() {
		SessionLocaleResolver
localeResolver= new
SessionLocaleResolver();
localeResolver.setDefaultLocale(Locale.US);
	}
}
```

SpringRestApplication

```
@Bean
 public ResourceBundleMessageSource
resourceBundleMessageSource() {
	ResourceBundleMessageSource
resourceBundleMessageSource= new
ResourceBundleMessageSource();
  resourceBundleMessageSource.setBasena
me("messages");
  return resourceBundleMessageSource;
}
```

Property Files

files

- messages.properties
  - good.morning.message=Hello World
- messages_fr.properties
  - good.morning.message=Bonjour le monde

HelloWorldController

```
@GetMapping(path= "/hello")
public String
helloWorld(@RequestHeader(name="Acce
pt-Language", required = false) Locale
locale) {
        return
messageSource.getMessage("good.mornin
g.message", null, locale);
  }
```

Spring Simplified

```
@GetMapping(path= "/hello")
public String helloWorld() {
	return messageSource.getMessage("good.morning.message",
null, LocaleContextHolder.getLocale());
}
```

```
@Bean
public LocaleResolver localeResolver() {
	AcceptHeaderLocaleResolver localeResolver= new AcceptHeaderLocaleResolver();
	localeResolver.setDefaultLocale(Locale.US);
	return localeResolver;
}
```

Or spring boot config

```
spring.messages.basename=messages
```

CONTENT NEGOTIATION

What does our application accept

- When we state nothing what are we getting?
- When we set Accept to application/json what do we get?
- When we set Accept to application/xml what do we get?
- To solve this add support for

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-xml -->
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

DEVELOPER TOOLS

Configure devtools

- XML

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

- IntelliJ
- Compiler
  - File > Settings > Build, Execution, Deployment > Compiler
    - Build project automatically
- Advanced Settings
  - File > Settings > Advanced Settings
    - Allow auto-make to start even if developed application is currently running

PROFILES

Steps to profiling

- Create the profile files
- Select the profile file from application.properties

- spring.profiles.active=dev

- Or use vm params
- -Dspring.profiles.active=prod

files

application-dev.properties

- logging.level.org.springframework= debug
- spring.jpa.show-sql=true
- spring.h2.console.enabled=true
- spring.datasource.url=jdbc:h2:mem:testdb
- spring.datasource.driverClassName=org.h2.Driver
- spring.datasource.username=sa
- spring.datasource.password=password

application-prod.properties

- spring.h2.console.enabled=false
- spring.datasource.url=jdbc:mysql://localhost:3306/cs544db
- spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
- spring.datasource.username=ea
- spring.datasource.password=cs544
- server.error.include-stacktrace=never

SWAGGER

What is Swagger

- REST has no standard, no mandatory definition
- Swagger
  - Documentation format for REST services
  - A way for you to provide documentation to your users
- Xml

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>3.0.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>3.0.0</version>
</dependency>
```

Swagger Configuration

- @Configuration
- @EnableSwagger2
- public class SwaggerConfig {

- 	@Bean
- 	public Docket api() {
- 		return new Docket(DocumentationType.SWAGGER_2);
- 	}
- }