Add instructor notes here.

Objective

- What is swagger
- Swagger-Basic Structure
- Implementing Swagger with Spring Boot Micro Services
- Looger with Spring boot
- Implementing Logger in Spring Boot Micro Services

Swagger

- **Swagger** is an open-source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful web services.
- Users identify Swagger by the Swagger UI tool, the Swagger toolset includes support for automated documentation, code generation, and test-case generation.
- Swagger allows you to describe the structure of your APIs so that machines can read them. The ability of APIs to describe their own structure is the root of all awesomeness in Swagger.
- Swagger's open-source tooling usage can be broken up into different use cases :
  - development
  - interaction with APIs
  - documentation.
- The Swagger API project was made open source in September 2011.

Swagger allows you to describe the structure of your APIs so that machines can read them. The ability of APIs to describe their own structure is the root of all awesomeness in Swagger. Why is it so great? Well, by reading your API's structure, we can automatically build beautiful and interactive API documentation. We can also automatically generate client libraries for your API in many languages and explore other possibilities like automated testing. Swagger does this by asking your API to return a YAML or JSON that contains a detailed description of your entire API. This file is essentially a resource listing of your API which adheres to OpenAPI Specification.
The specification asks you to include information like:
What are all the operations that your API supports?
What are your API's parameters and what does it return?
Does your API need some authorization?
And even fun things like terms, contact information and license to use the API.

Swagger

Usage of Swagger :

- **Developing of API**
- **Interacting with API**
- **Documenting API**

Developing APIs

When creating APIs, Swagger tooling may be used to automatically generate an Open API document based on the code itself. This is informally called code-first or bottom-up API development. While the software code itself can accurately represent the Open API document, many API developers consider this to be an outdated technique as it embeds the API description in the source code of a project and is typically more difficult for non-developers to contribute to.

Alternatively, using Swagger Codegen developers can decouple the source code from the Open API document, and generate client and server code directly from the design. While considered complicated, this has been considered a more modern API workflow by many industry experts and allows more freedom when designing the API by deferring the coding aspect.

Interacting with APIs

Using the Swagger Codegen project, end users generate client SDKs directly from the OpenAPI document, reducing the need for human-generated client code. As of August 2017, the Swagger Codegen project supported over 50 different languages and formats for client SDK generation.

Documenting APIs

When described by an OpenAPI document, Swagger open-source tooling may be

used to interact directly with the API through the Swagger UI. This project allows connections directly to live APIs through an interactive, HTML-based user interface. Requests can be made directly from the UI and the options explored by the user of the interface.

Basic Structure-Swagger

Swagger definitions can be written in JSON or YAML.

Metadata

Every Swagger specification starts with the Swagger version, 2.0 being the latest version. A Swagger version defines the overall structure of an API specification – what we can document and how we document it.


```
{ "swagger": "2.0", "info": { "description": "Api Documentation", "version": "1.0", "title": "Api Documentation", "termsOfService": "urn:tos", "contact": {}, "license": { "name": "Apache 2.0", "url": "http://www.apache.org/licenses/LICENSE-2.0" } },
```


Base URL

The base URL for all API calls is defined using schemes, host and basePath:All API paths are relative to the base URL. For example, /users actually means https://api.example.com/v1/users.


```
"host": "localhost:9090", "basePath": "/", "tags": [ { "name": "basic-error-controller", "description": "Basic Error Controller" }, { "name": "product-controller", "description": "Product Controller" } ],
```


In Post check swagger
http://localhost:9090/v2/api-docs

REST APIs have a base URL to which the endpoint paths are appended. The base URL is defined by schemes, host and basePath on the root level of the API specification. All API paths are relative to this base URL, for example, /users actually means <scheme>://<host>/<basePath>/users.
schemes are the transfer protocols used by the API. Swagger supports the http, https, and WebSocket schemes – ws and wss.
host is the domain name or IP address (IPv4) of the host that serves the API. It may include the port number if different from the scheme's default port (80 for HTTP and 443 for HTTPS). basePath is the URL prefix for all API paths, relative to the host root. It must start with a leading slash /. If basePath is not specified, it defaults to /, that is, all paths start at the host root.
An API can accept and return data in different formats, the most common being JSON and XML. We can use the consumes and produces keywords to specify the MIME types understood by your API. The value of consumes and produces is an array of MIME types. Global MIME types can be defined on the root level of an API specification and are inherited by all API operations.

Basic Structure-Swagger

Paths:

- The paths section defines individual endpoints (paths) in your API, and the HTTP methods (operations) supported by these endpoints.

```
"paths": {       "/": {          "get": {             "tags": [
"product-controller"             ],             "summary": "getAll",
"operationId": "getAllUsingGET",             "produces": [
"*/*"             ],             "responses": {                "200": {
"description": "OK",                    "schema": {
"type": "array",                   "items": {
"$ref": "#/definitions/Product"                        }                }
},
```

- The consumes and produces sections define the MIME types supported by the API. The root-level definition can be overridden in individual operations.
- For each operation, we can define possible status codes, such as 200 OK or 404 Not Found, and schema of the response body. Schemas can be defined inline or referenced from an external definition via $ref.

Basic Structure-Swagger

Authentication

The securityDefinitions and security keywords are used to describe the authentication methods used in our API.

Input and Output Models

The global definitions section lets us define common data structures used in our API. They can be referenced via $refwhenever a schema is required – both for request body and response body.

Parameters

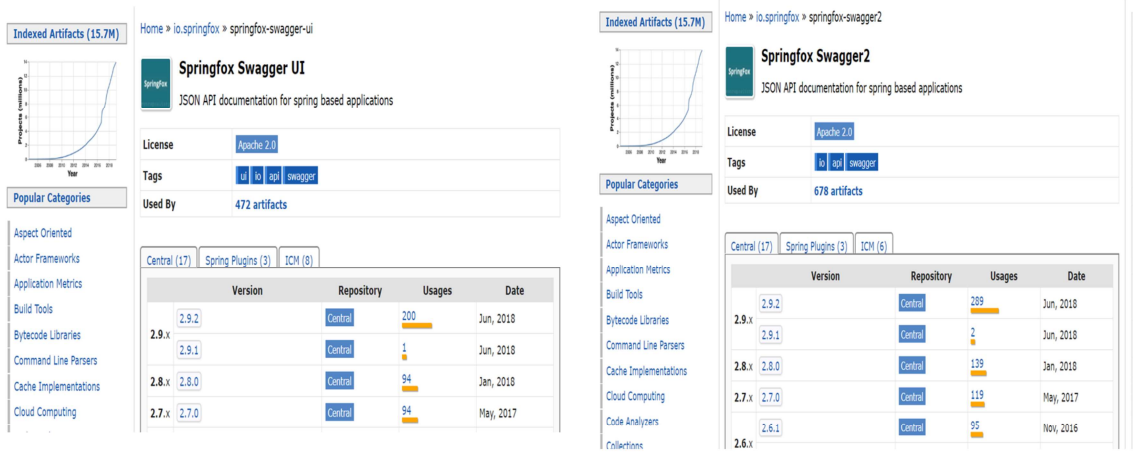
Operations can have parameters that can be passed via URL path (/users/{userId}), query string (/users?role=admin), headers (X-CustomHeader: Value) and request body.

Adding Swagger in Spring Boot

Steps:

1. **Getting Swagger Spring Dependency**
2. **Enabling Swagger in our Code**
3. **Configuring Swagger**
4. **Adding Details as Annotation to API**

Getting Swagger Spring Dependency



Getting Swagger Spring Dependency-Maven

Springfox Swagger UI

```
<!--
https://mvnrepository.com/artifact/
io.springfox/springfox-swagger-ui  -
->
<dependency>

<groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-
ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

Springfox Swagger2

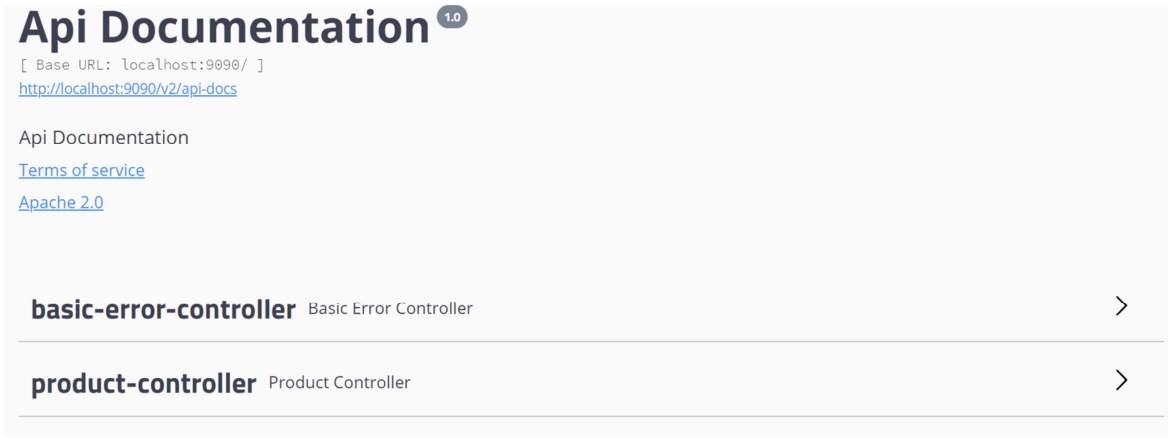
```
<!--
https://mvnrepository.com/artifact/io.spri
ngfox/springfox-swagger2 -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-
swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
```

Enabling Swagger in our Code

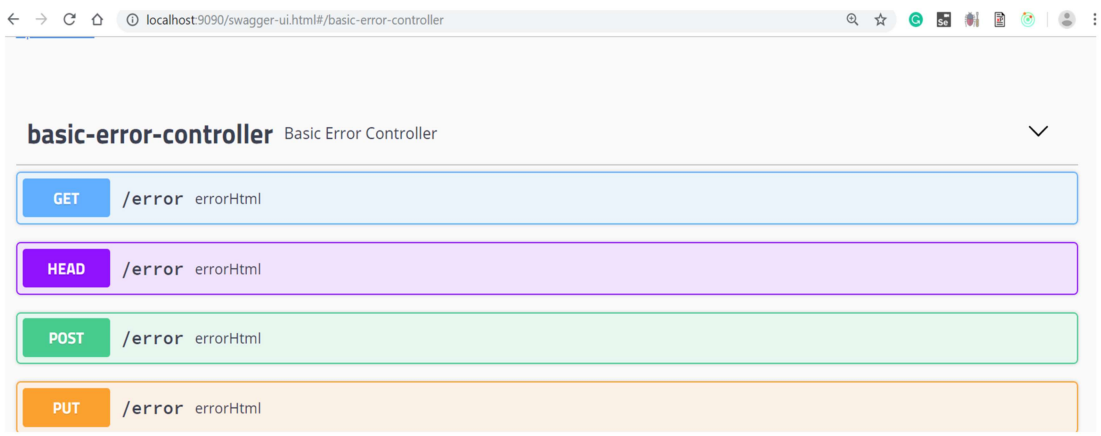
@EnableSwagger2 - Annotation to Enable Swagger Documentation on the API

```
1 package com.cg.SwaggerBasic;
2
3 import org.springframework.boot.SpringApplication;
7
8 @SpringBootApplication
9 @EnableSwagger2
10 public class SwaggerBasicApplication {
11
12     public static void main(String[] args) {
13         SpringApplication.run(SwaggerBasicApplication.class, arg
14     }
15
16 }
```

Swagger-UI

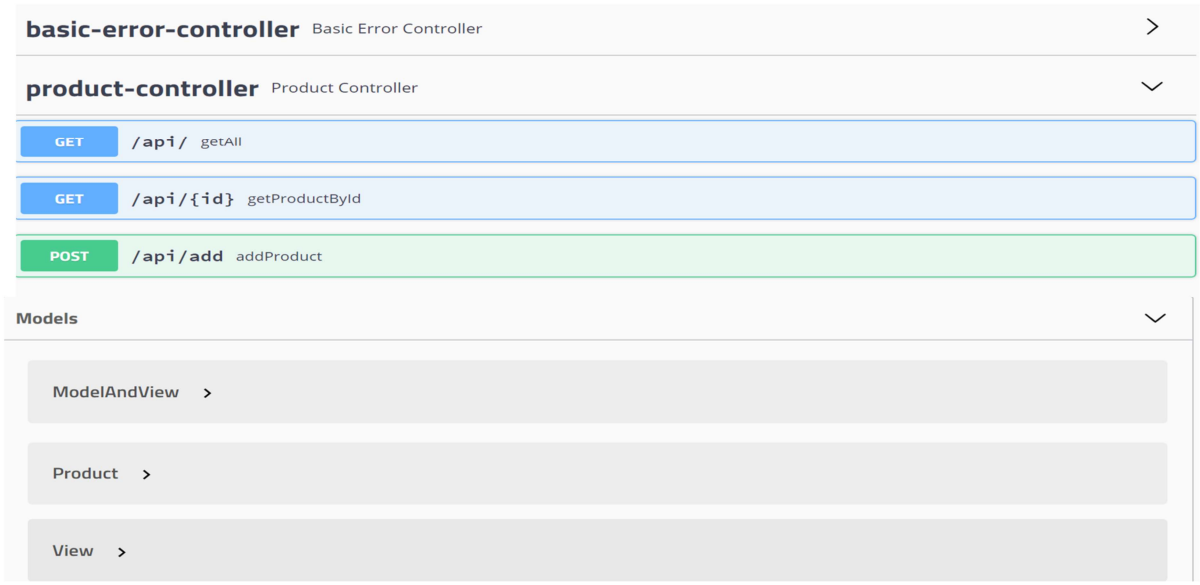

Api Documentation
1.0
[Base URL: localhost:9090/]
http://localhost:9090/v2/api-docs
Api Documentation
Terms of service
Apache 2.0
basic-error-controller Basic Error Controller
product-controller Product Controller

Swagger-UI



http://localhost:9090/swagger-ui.html

Swagger-UI


basic-error-controller Basic Error Controller
product-controller Product Controller
GET /api/ getAll
GET /api/{id} getProductById
POST /api/add addProduct
Models
ModelAndView
Product
View

Docket

- @EnableSwagger2 annotation which indicates that Swagger support should be enabled.
- A Docket bean in a Spring Boot configuration to configure Swagger 2 for the application. A Springfox Docket instance provides the primary API configuration with sensible defaults and convenience methods for configuration.
- PathSelectors provides additional filtering with predicates which scan the request paths of your application. We can use any(), none(), regex(), or ant().
- Swagger's response by passing parameters to the ***apis()*** and ***paths()*** methods of the *Docket* class.
- We can use the apiInfo(ApiInfo apiInfo) method. The ApiInfo class that contains custom information about the API.

Why adding Docket:
All the error pages documentation & other documentation which is not useful is omitted .
API documentation Page should be replaced by who is User, so we are using Docket

Swagger Core annotation

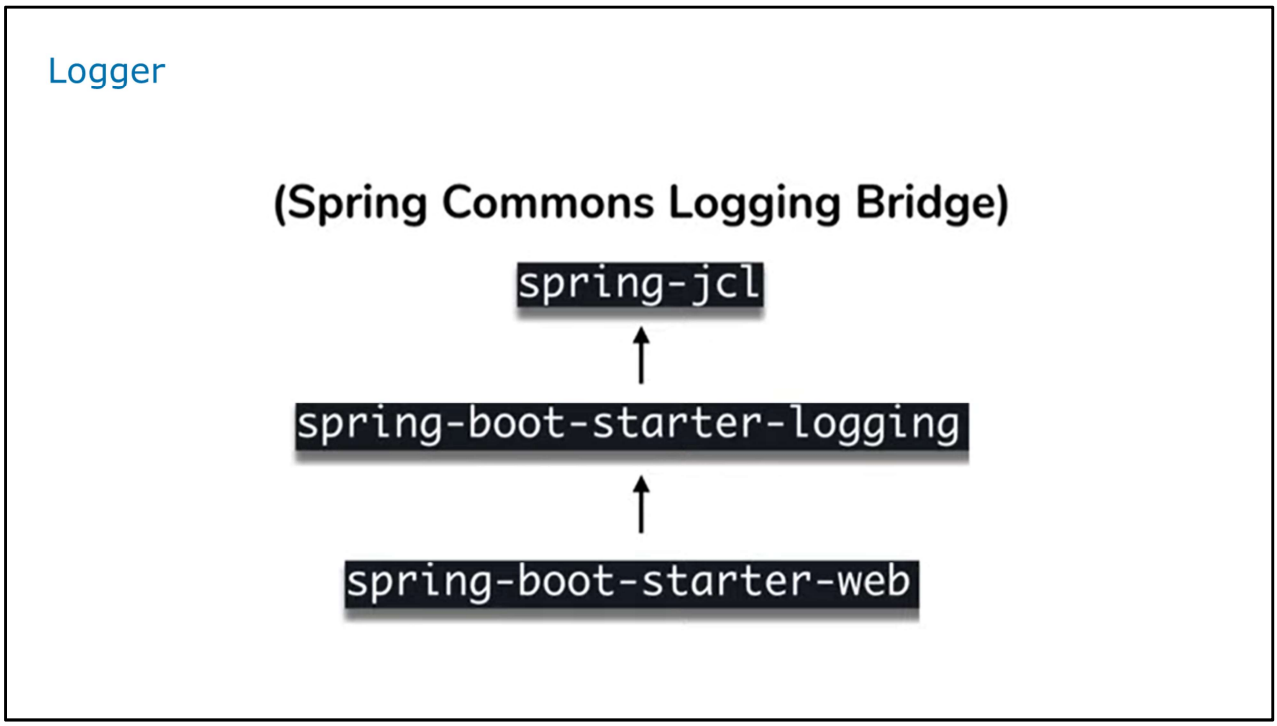
We can use the @Api annotation on our ProductController class to describe our API.

@Api	Marks a class as a Swagger resource.
@ApiModel	Provides additional information about Swagger models.
@ApiModelProperty	Adds and manipulates data of a model property.
@ApiOperation	Describes an operation or typically an HTTP method against a specific path.
@ApiParam	Adds additional meta-data for operation parameters.
@ApiResponse	Describes a possible response of an operation.
@ApiResponses	A wrapper to allow a list of multiple ApiResponse objects.

Demo

SwaggerBasic

Logger



Spring Boot is a very helpful framework — it allows us to forget about the majority of the configuration settings, many of which it opinionatedly autotunes.
In the case of logging, the only mandatory dependency is *Apache Commons Logging.* We need to import it only when using Spring 4.x (Spring Boot 1.x), since in Spring 5 (Spring Boot 2.x) it's provided by Spring Framework's ***spring-jcl*** module.
We shouldn't worry about importing *spring-jcl* at all if we're using a Spring Boot Starter (which almost always we are). That's because every starter, like our *spring-boot-starter-web*, depends on *spring-boot-starter-logging,* which already pulls in *spring-jcl* for us.
When using starters, Logback is used for logging by default.
Spring Boot pre-configures it with patterns and ANSI colors to make the standard output more readable.

Logger

Application.properties

```
#looger
logging.level.root=WARN
logging.level.com.cg=INFO
```

Java Code

```
Logger logger =
LoggerFactory.getLogger(LoggingController.class);
```

Demo

DemoH2Database