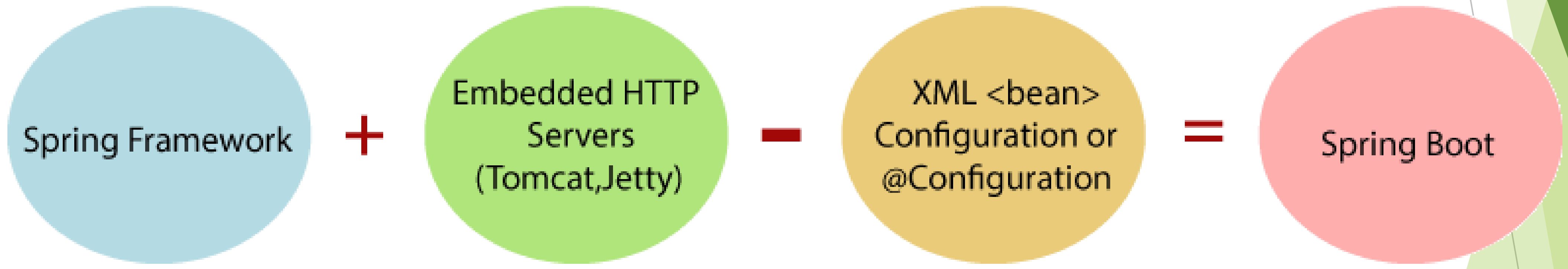
Spring Boot

dasturlash.uz

What is Spring Boot

- Spring Boot is a project that is built on the top of the Spring Framework.
- It provides an easier and faster way to set up, configure, and run both simple and web-based applications.
- It is a Spring module that provides the **RAD (*Rapid Application Development*)** feature to the Spring Framework.
- It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.
- Spring Boot bu Spring Framework ustiga qurilgan loyiha dir.
- Spring boot Springa asoslangan oddiy yokiy web projectlarni oson va tez konfiguratsiya qilib berdi.

Spring boot



Spring Framework
+
Embedded HTTP Servers (Tomcat,Jetty)
-
XML <bean> Configuration or @Configuration
=
Spring Boot

dasturlash.uz

Spring Boot

- In short, Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.
- In Spring Boot, there is no requirement for XML configuration (deployment descriptor).
- It uses convention over configuration software design paradigm that means it decreases the effort of the developer.
- We can use Spring **STS IDE** or **Spring Initializr** to develop Spring Boot Java applications.
- Qisqa qilib aytsak Spring Boot bu Spring Framework va Embedded Server larni birikmasidir.
- Spring Bootda XML konfig larni yozib o’tirmaymiz.

Spring Boot

- **Why should we use Spring Boot Framework?**

- We should use Spring Boot Framework because:
  - The dependency injection approach is used in Spring Boot.
  - It contains powerful database transaction management capabilities.
  - It simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts, etc.
  - It reduces the cost and development time of the application.

Spring Boot

- Along with the Spring Boot Framework, many other Spring sister projects help to build applications addressing modern business needs. There are the following Spring sister projects are as follows:
- ***Spring Data***: It simplifies data access from the relational and **NoSQL** databases.
- ***Spring Batch***: It provides powerful **batch** processing.
- ***Spring Security:*** It is a security framework that provides robust **security** to applications.
- ***Spring Social***: It supports integration with **social networking** like LinkedIn.
- ***Spring Integration***: It is an implementation of Enterprise Integration Patterns. It facilitates integration with other **enterprise applications** using lightweight messaging and declarative adapters.

dasturlash.uz

Spring Boot

- Create Spring boot project
  - https://start.spring.io/

dasturlash.uz

Spring Boot

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

Rest

- Rest - Dam olish   deb tarjima qilinadi.
- **REST** stands for *REpresentational State Transfer*, a standardized approach to building web services.
- REST is an **architectural approach.**
- REST bu *REpresentational State Transfer* ning qisqartmasi. Yani REST bu serverlarni yaratishda standartlashtirilgan yondashuv deyiladi.
-

REST API

- A **REST API** is an intermediary *Application Programming Interface* that enables two applications to communicate with each other over HTTP, much like how servers communicate to browsers.
- RESTful is the most common approach for building web services because of how easy it is to learn and build.
- A REST API (also known as RESTful API)
- REST API bu ikkita serverni bir biriga URL bilan bog’laydigan qatlam (dasturiy taminot)
- RESTful - bu web server larni yaratishda eng keng tarqalgan yondashuv, chunki uni o'rganish va ishlash juda oson.

dasturlash.uz

RESTful

- Advantages of RESTful web services
  - RESTful web services are **platform-independent**.
  - It can be written in any programming language and can be executed on any platform.
  - It provides different data format like **JSON, text, HTML,** and **XML**.
  - It is fast in comparison to SOAP because there is no strict specification like SOAP.
  - These are **reusable**.
  - They are **language neutral**.

Request and Response

- Request – so’rov, murojat
- Response – javob, natija

HTTP Methods

- GET: It reads a resource.
- PUT: It updates an existing resource.
- POST: It creates a new resource.
- DELETE: It deletes the resource.

Rest Api example

- **POST /users:** It creates
- **GET /users/{id}:** It retrieves the detail of a user. a user.
- **GET /users:** It retrieves the detail of all users.
- **DELETE /users:** It deletes all users.
- **DELETE /users/{id}:** It deletes a user.

Spring WEB

- Spring WEB – module enable Spring Rest Api .

dasturlash.uz

@RestController

- Spring 4.0 introduced the *@RestController* annotation in order to simplify the creation of RESTful web services.
- **It's a convenient annotation that combines *@Controller* and *@ResponseBody***, which eliminates the need to annotate every request handling method of the controller class with the *@ResponseBody* annotation.

@RequestMapping

- Annotation is used to map web requests to Spring Controller methods.

```
@RequestMapping(value = "/ex/foos", method = RequestMethod.GET)

public String getFoosBySimplePath() {
            return "Get some Foos";
  }
```

http://localhost:8080/spring-rest/ex/foos

@RequestMapping Consumes and Produces

- Mapping **media types produced by a controller** method is worth special attention.
- We can map a request based on its *Accept* header via the *@RequestMapping headers* attribute introduced above:

```
@RequestMapping( value = "/ex/foos", method = GET, headers = "Accept=application/json")
public String getFoosAsJsonFromBrowser() {
            return "Get some Foos with Header Old";
}
```

@GetMapping

- The @GetMapping annotation is a specialized of @RequestMapping annotation that acts as a shortcut for :
  - @RequestMapping(method = RequestMethod.GET).

- The @GetMapping annotated methods in the *@Controller* annotated classes handle theHTTP GET requests matched with given URI expression.

@PostMapping

- The *@PostMapping* is specialized version of @RequestMapping annotation that acts as a shortcut for
  - @RequestMapping(method = RequestMethod.POST).
- The @PostMapping annotated methods in the *@Controller* annotated classes handle the HTTP POST requests matched with given URI expression.

@RequestMapping short hands

- @GetMapping
- @PostMapping
- @PutMapping
- @DeleteMapping
- @PatchMapping

@PathVariable

- **The *@PathVariable* annotation can be used to handle template variables in the request URI mapping**, and set them as method parameters.

-
  ```
  @GetMapping("/api/employees/{id}")
  public String getEmployeesById( @PathVariable String id ) {
          return "ID: " + id;
  }
  ```

Multiple Path Variables in a Single Request

- **Multiple Path Variables in a Single Request**
- Depending on the use case, **we can have more than one path variable in our request URI for a controller method, which also has multiple method parameters:**
-

```
@GetMapping("/api/employees/{id}/{name}")
public String getEmployeesByIdAndName(@PathVariable String id, @PathVariable String name) {
  return "ID: " + id + ", name: " + name;
}
```

dasturlash.uz

@PathVariable Links

- https://www.baeldung.com/spring-pathvariable#2-using-javautiloptional

dasturlash.uz

@RequestParam

- **We can use *@RequestParam* to extract query parameters, form parameters, and even files from the request.**

- Let's say that we have an endpoint */api/foos* that takes a query parameter called *id*:

-
  ```
  @GetMapping("/api/foos")
  public String getFoos(@RequestParam(“id”) String id) {
    return "ID: " + id;
  }
  ```

- http://localhost:8080/spring-mvc-basics/api/foos?id=abc

@RequestParam links

- https://www.baeldung.com/spring-request-param

ResponseEntity

ResponseEntity

- *ResponseEntity* **represents the whole HTTP response: status code, headers, and body**.
- As a result, we can use it to fully configure the HTTP response.
- If we want to use it, we have to return it from the endpoint; Spring takes care of the rest.

ResponseEntity - Example

```
 @GetMapping("/hello")
  public String hello() {
      return "Hello World";
 }

@GetMapping("/hello")
 public ResponseEntity<String> hello() {
     return new ResponseEntity<>("Hello World!", HttpStatus.OK);
 }


 @GetMapping("/hello")
 public ResponseEntity<String> hello() {
     ResponseEntity<String> responseEntity = new ResponseEntity<>("Hello World!", HttpStatus.OK);
     return  responseEntity;


 }
```

dasturlash.uz

```java
public class HttpEntity<T> {
  private final HttpHeaders headers;
  private final T body;
}


public class ResponseEntity<T> extends HttpEntity<T> {
   private final Object status;
}
```

- For the most popular HTTP status codes we get static methods:
- BodyBuilder **accepted**();
- BodyBuilder **badRequest**();
- BodyBuilder **created**(java.net.URI location);
- HeadersBuilder<?> noContent(); HeadersBuilder<?> notFound();
- BodyBuilder **ok**();

Example

```
@GetMapping("/age")
 ResponseEntity<String> age(@RequestParam("yearOfBirth") int yearOfBirth) {

  if (isInFuture(yearOfBirth)) {
            return ResponseEntity.badRequest()
                          .body("Year of birth cannot be in the future");
  }

  return ResponseEntity.status(HttpStatus.OK)
                     .body("Your age is " + calculateAge(yearOfBirth));

  }
```

Custom Header

```
@GetMapping("/customHeader")
 ResponseEntity<String> customHeader() {

    return ResponseEntity.ok()
                        .header("Custom-Header", "foo")
                        .body("Custom header set");

}
```

Manipulate the Response Directly

- Spring also lets us access the *javax.servlet.http.HttpServletResponse* object directly; we only have to declare it as a method argument:
- @GetMapping("/manual") **void manual**(HttpServletResponse response) **throws** IOException {

```
  response.setHeader("Custom-Header", "foo");
  response.setStatus(200);
  response.getWriter().println("Hello World!");

}
```

Http Status Codes

dasturlash.uz

HTTP Status Codes

- HTTP status codes are divided into 5 classes based on the first digit of the status code:
  - *1xx* – informational – the request was received, continuing process
  - *2xx* – successful – the request was received successfully and accepted
  - *3xx* – redirection – further action needs to be taken to complete the request
  - *4xx* – client error – request contains bad syntax or cannot be fulfilled
  - *5xx* – server error – the server failed to fulfill an apparently valid request

- https://www.baeldung.com/cs/http-status-codes

Http status - most used

- **200 OK**
  - A 200 status code indicates that the request was successful and could be the result of any of the following operations: GET, POST, PUT, PATCH, and DELETE.
- **201 Created**
  - A 201 response indicates that a request was successful and resulted in a new resource being created. An example of a request which could result in a 201 status code is when we submit a form to create a new resource.
- **301 Found**
  - A 301 response tells us that the resource we're requesting has moved permanently.

Http status - most used

- **400 Bad Request**
  - This status code means that the HTTP request sent over to the server has incorrect syntax
  - For example, if we have submitted a form with incorrectly formatted data or the wrong data type, this could result in a 400 response.
- **401 Unauthorized**
  - This means that the user who is trying to access the resource by sending the request has not been properly authenticated
  - According to the HTTP standards, when a server responds with a 401 it must also send a *WWW-Authenticate* header with a list of the authentication schemes that the server uses.
- **403 Forbidden**
  - A 403 response is commonly confused with 401; however, the 403 response means that the server is refusing our request due to insufficient access or permissions.
  - Even if we’re authenticated, we may not be allowed to make a certain request.

Http status - most used

- **404 Not Found**
  - This status code is returned when the requested resource is not found
  - For instance, if we entered a URL for a page that does not exist on a website or was deleted for some reason, it would result in a 404 response.
- **405 Method Not Allowed**
  - A 405 response means that the server does not support the method used in the request
  - For example, if an endpoint only accepts GET requests and we try to send a POST, it will result in a 405 response code. If this is the case, servers must also send back an *Allow* header field with the list of the allowed methods.
- **408 Request Timeout**
  - This status code is an indication from the server that the request message was not completed within the time that the server was prepared to wait
  - If a client is too slow in sending its HTTP request, this could happen. This could be due to many reasons, including a slow internet connection.

Http status - most used

- **413 Request Entity Too Large**
  - This status code tells us that the server is unable or unwilling to process the request because the request payload is too large. This could happen if we try to upload a file that is too large and exceeds the server limits.
- **500 Internal Server Error**
  - An internal server error means that for some reason the server could not process the request due to an error. This is generally due to a bug or an unhandled exception on the server.

Http status - most used

- **Unassigned HTTP Status Codes**
  - HTTP status codes are extensible, and the ones we listed above are only a small part of the many status codes in use.
- 104 to 199
  209 to 225
  227 to 299
  309 to 399
  418 to 420
  427
  430
  432 to 450
  452 to 499
  509
  512 to 599