Creating a Spring Boot Application

Using Spring Initializr

Web Applications

- We will use Spring Boot to create a web application
- Web applications use HTTP as a protocol for communicating over the internet
- Devices connected to the internet find each other by their unique public IP adresses
- We will create local web applications with IntelliJ they will be found by the internal IP address 127.0.0.1 also known as localhost

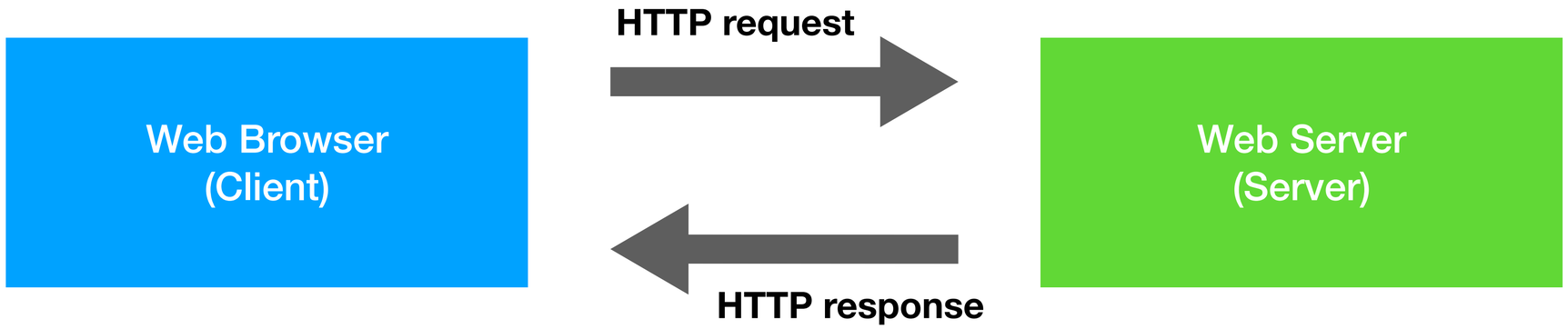
Ports

- There could be many local applications on a computer, and the local IP number to the local machine would be used by all of them
- Therefore the computer also has 65536 ports and each application occupies one port each
- The standard port used by a Spring Boot web application is 8080 (we can change this easily if we want to)
- So this is the web address to our application: http://localhost:8080

HTTP (HyperText Transmission Protocol

- HTTP is the protocol we use to send requests and responses over the web

- HTTP requests and responses are plain text



HTTP is stateless

- HTTP is stateless - the server doesn’t remember previous requests
- Every request is independent of other requests
- The server doesn’t know if requests are coming from the same client or different clients

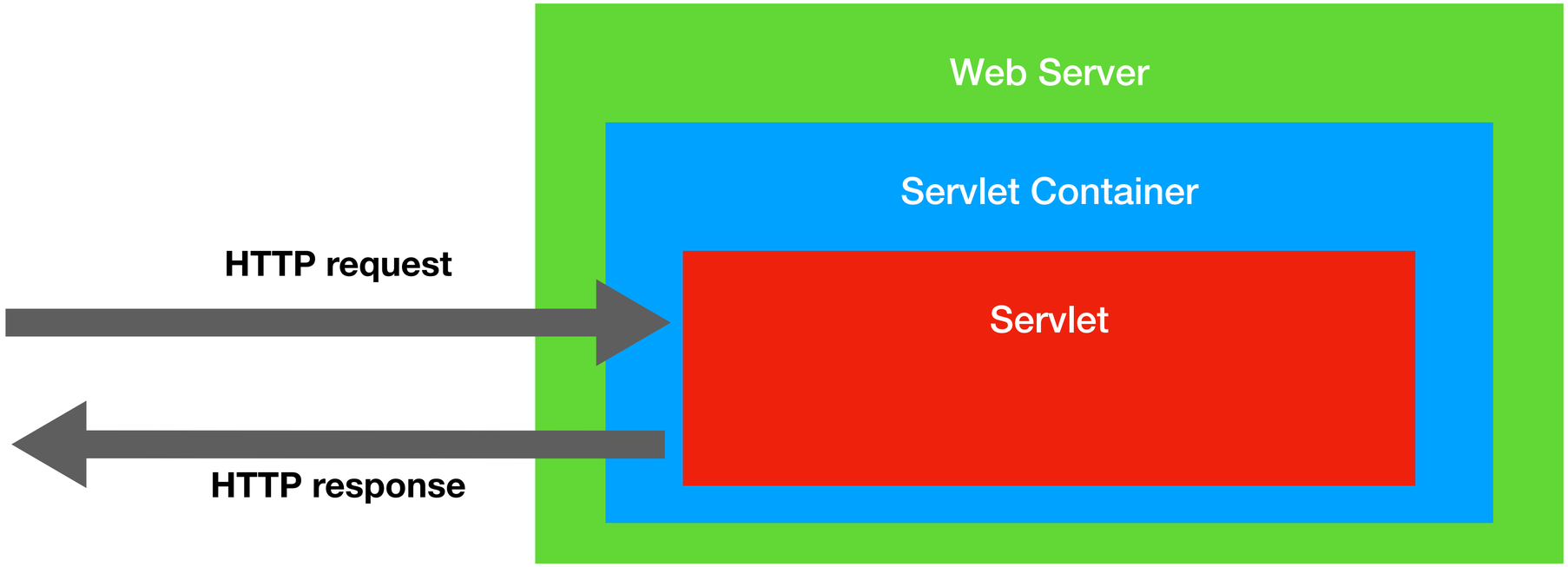
HTTP methods

- GET - Gets a resource from the server (with no side effects)
- POST - Create a new resource on the server (sends data to the server)
- PUT - Update an existing resource on the server (sends data to the server)
- DELETE - Deletes a resource from the server

HTTP and Web browsers

- A web browser renders web pages created with HTML
- HTML in a web browser can only trigger GET and POST requests
- Links and a URL in the address bar triggers a GET request
- Forms in HTML can trigger GET or be set to use POST instead

Java Web Applications



Servlet

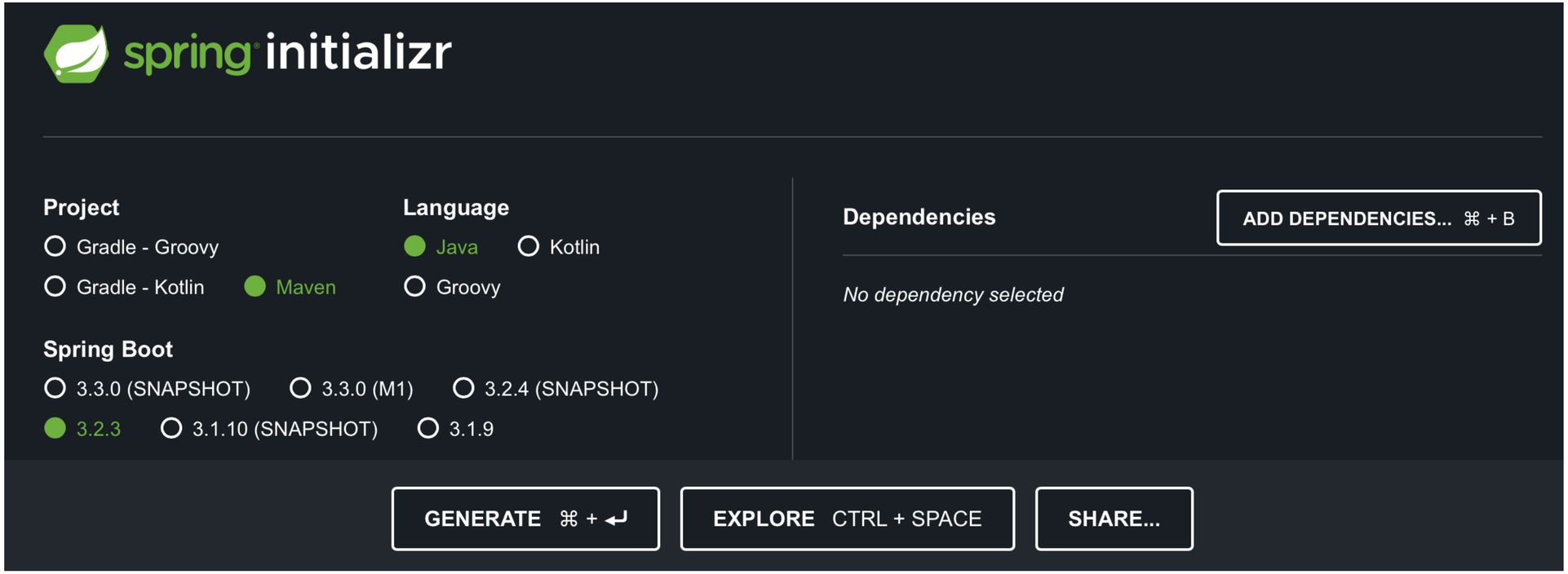
- The Servlet is a Java class extending from HttpServet
- A doGet method in the servlet handles all GET requests
- A doPost method in the servlet handles all POST requests
- These methods get two input arguments, a HttpServletRequest object and a HttpServletResponse object

A Spring Boot web application

- We will use Spring Boot to create a web application
- Spring handles the Servlet and makes it possible to handle web requests in an easier way with methods in Controller classes
- We will dive right into Spring Boot!

Using Spring Initializr

- A Spring Boot application can be generated at the Spring Initializr
- Spring Initializr can be found at https://start.spring.io



Demo 1 - Spring Boot web application

- Create a Spring Boot project on https://start.spring.io
- Select Maven under Project, click ADD DEPENDENCIES, select Web, then click Generate to download the project
- Create a Controller class next to the …Application class
- Create a @GetMapping method in the Controller
- Test the web application from a web browser

Handling requests

- The request to a Controller method will contain a URL and possibly:
- Request parameters after the URL
- Path variables in the URL

Request parameters

- Request parameters are sent after the URL, seperated with a question mark, the name of the parameter an equals sign and the value, like this request parameter with the name param1 and the value hello

- http://localhost:8080?param1=hello

- Input argument should look like this: @RequestParam String param1

Path variables

- Path variables are sent as part of the URL, and mapped within curly braces to receive the value as a parameter
- http://localhost:8080/ex4/hello
- GetMapping should look like this: @GetMapping("/ex4/{name}")
- Input argument should look like this: @PathVariable String name

Demo 2 - Request params and Path Variables

- Request params (required, not required, default value)
- Path variable

Exercise 1 - Spring Boot application

- Create a Spring Boot web application on https://start.spring.io
- Create a Controller class annotated with @RestController
- Inside the class, create a method with a @GetMapping method, like this:

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

- Try it out using a web browser with the URL http://localhost:8080

Exercise 2 - Request params and Path Variables

- 1. Create a @GetMapping method that takes two int values as input and return the sum of the input arguments, using request params
- 2. Create another @GetMapping method that takes two int values as input and return the sum of the input arguments, using path variables
- Several path variables can be used in a URL with a / in between
- Several request params can be used in a URL with a & in between
- Try it out using a web browser