Spring Boot workshop

for jaksiemasz.care

Goal

Create simple web application
“Shouter” using Spring Boot

Functionalities:

- registering users
- users authorization
- adding simple posts, named “Shouts” by users
- reading posts

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

spring
by Pivotal™

VS

spring
boot

Spring

Spring Boot

Spring Boot

Spring Boot advantages

- Auto-configuration
- Starter dependencies
- Almost ready to use
- The shortest hello world requires 1 file and 7 lines of code :O

Spring magic

Dependency Injection

When you need some componenets...

```
public class Foo {
  private PrintStream printStream;

  public Foo() {
    this.printStream = new PrintStream(System.out);
  }

  public void print() {
    printStream.println("I'm foo!");
  }
```

```
public class Foo2 {
  private PrintStream printStream;

  public Foo() {
    this.printStream = new PrintStream(System.out);
  }

  public void print() {
    printStream.println("I'm foo2!");
  }
```

Meh...

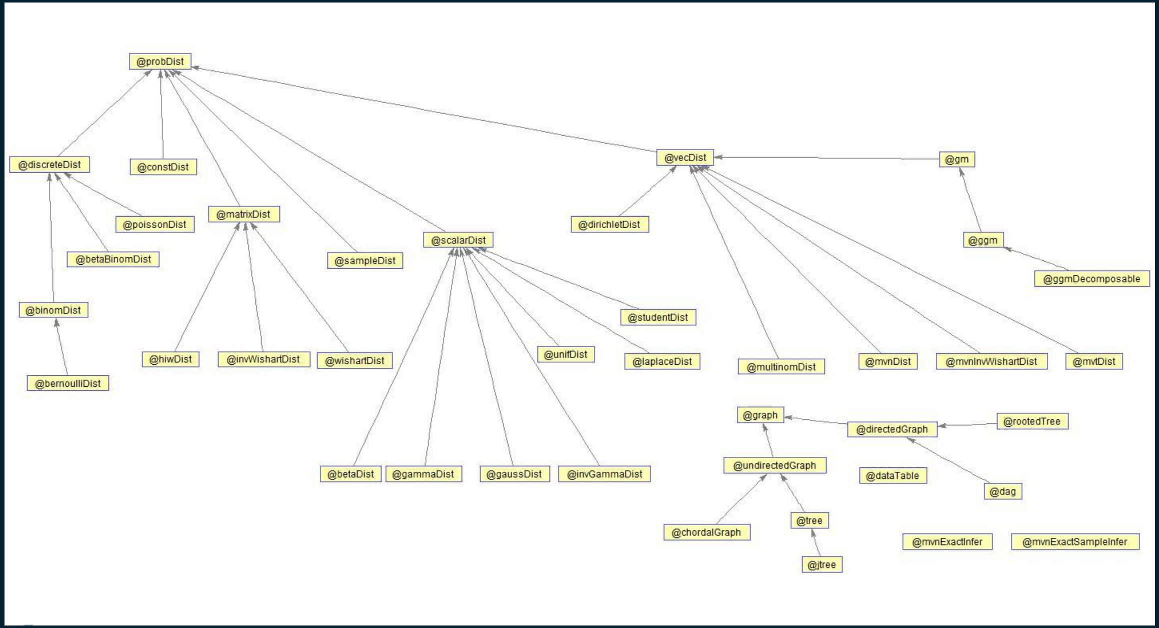
Inheritance?

```java
public class FooWithPrintStream {
   protected PrintStream printStream;

   public FooWithPrintStream() {
      this.printStream = new PrintStream(System.out);
   }

}
```

```java
public class Foo extends FooWithPrintStream {
   public void print() {
      printStream.println("I'm foo!");
   }
}
```

```java
public class Foo2 extends FooWithPrintStream {
   public void print() {
      printStream.println("I'm foo2!");
   }
}
```

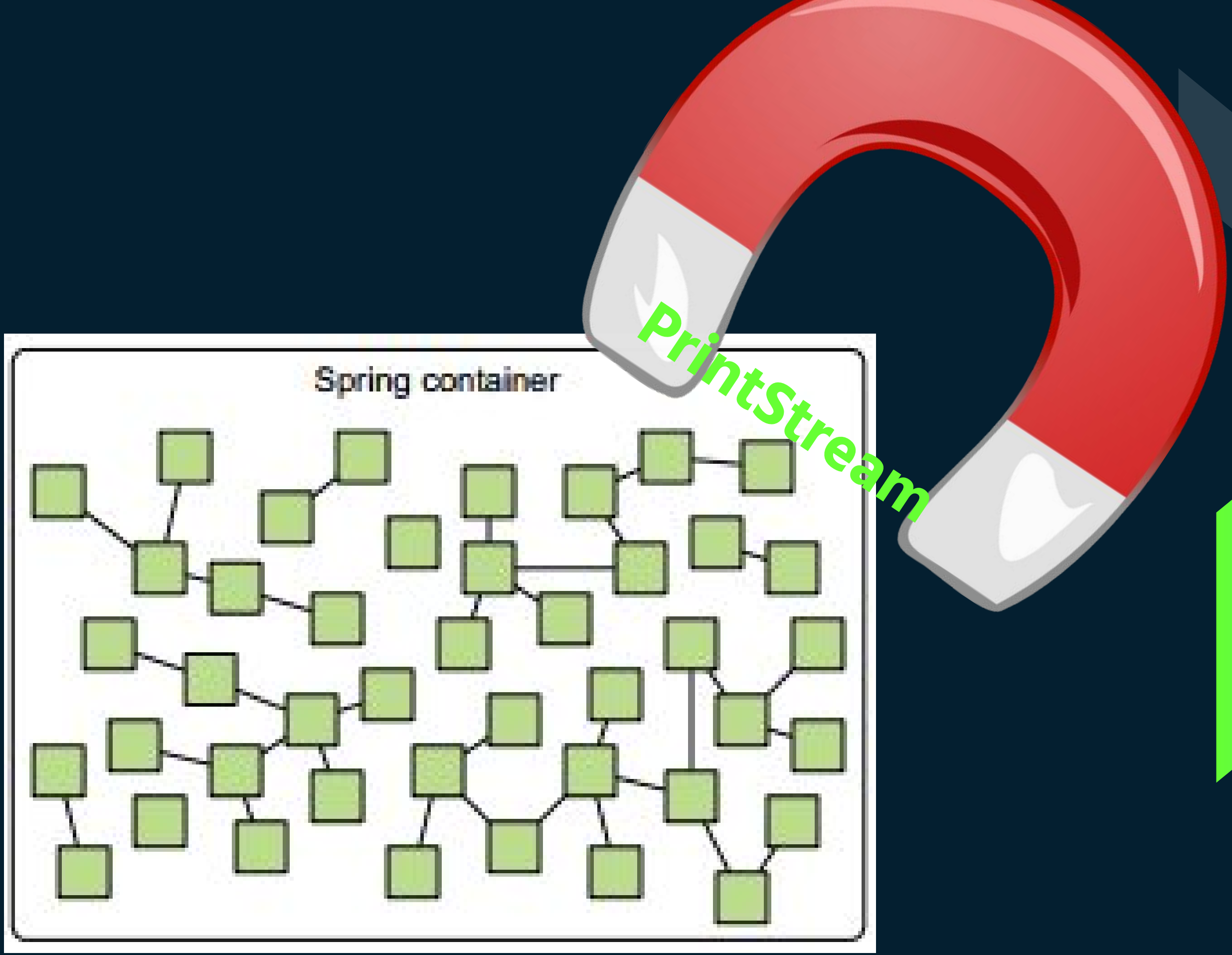
May end up with giant inheritance hierarchy... Meh



@Bean

“Declare once, use everywhere!”

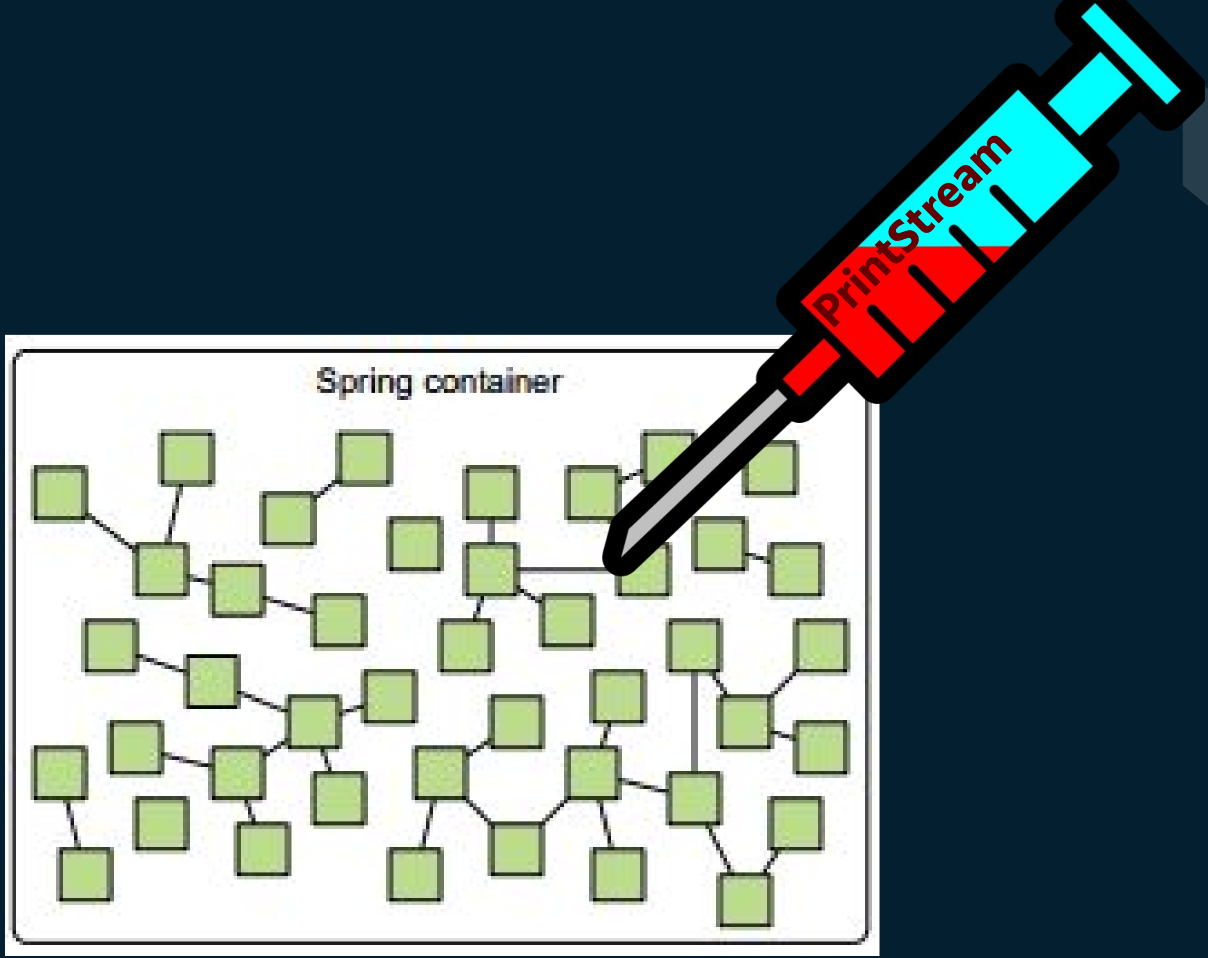
```
@Configuration
public class PrintingConfiguration {

  @Bean
  public PrintStream getPrintStream() {
     return new PrintStream(System.out);
  }

}
```


Spring container
PrintStream

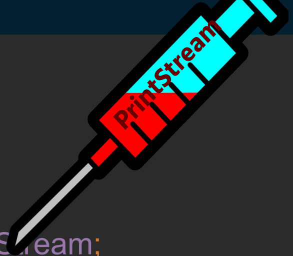
@Autowired

```java
public class Foo {

  @Autowired
  private PrintStream printStream;

  public void print() {
     printStream.println("I'm foo!");
  }
```

```java
public class Foo2 {

  @Autowired
  private PrintStream printStream;

  public void print() {
     printStream.println("I'm foo2!");
  }
```


Spring container
PrintStream

```
public class Foo {

   @Autowired
   private PrintStream printStream;

   public void print() {
       printStream.println("I'm foo!");
   }
```



```
public class Foo2 {

   @Autowired
   private PrintStream printStream;

   public void print() {
       printStream.println("I'm foo2!");
   }
```

Request mapping

Simple controller

```
@Controller
public class SimpleController {

  @RequestMapping(value = "/", method = RequestMethod.GET)
  public @ResponseBody String getSomeRequestsHere() {
     return "I just got your request!";
  }

}
```

@Controller

- indicates that annotated class is a controller
- declares annotated class as a spring component

@RequestMapping

- indicates that annotated method will be handling HTTP request on a given URL with a given HTTP method

@ResponseBody

- indicates that a method return value should be part of web response body

Simplify

@Controller vs @RestController

- almost no difference;

  in @RestController you don’t have to annotate method return value for each mapped method

Use @'*method* 'Mapping instead of @RequestMapping(method=...)

@GetMappping("/resource")

Vs

@RequestMapping(method = HttpMethod.GET, value = "resource")

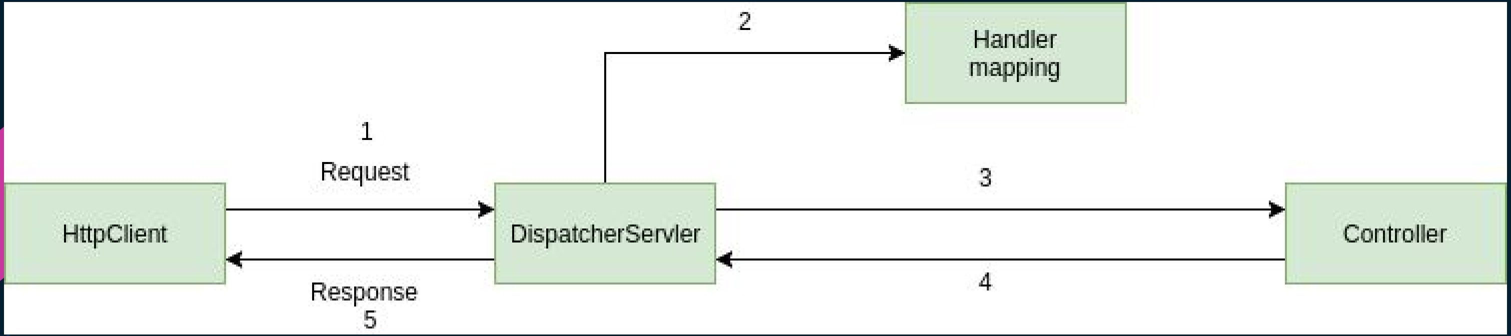
@PostMapping("/resource"

vs RequestMapping(method=HttpMethod.POST)

Simplified controller

```
@RestController
public class SimpleController {

  @GetMapping("/here")
  public String getSomeRequestsHere() {
     return "I just got your request!";
  }

}
```

OK… But how?


2
Handler mapping
1
Request
HttpClient
DispatcherServler
3
Controller
4
Response
5

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

Create Spring Boot project

https://start.spring.io

Or using IntellIJ

File → New->Project → Spring Initializr

Add starter dependencies

web - because we will build a web app

H2 - because we will need to start with an embedded database without any configuration

JPA - because we will need to persist data to database and it's quite nice and easy way to do this

Write hello world controller

controller.HelloWorldController

But how to use it?

- point your browser to http://localhost:8080/
- use Postman to send requests

or…

- write some tests!

Send request using Java

RestTemplate class offers methods to do that, e.g.:

```
restTemplate.getForObject("localhost:8080/resource", String.class);
```

performs GET on resource localhost... and tries to parse response into String.

Writing spring tests

@RunWith(SpringRunner.class)

- enables starting an application when test is performed

@SpringBootTest

- enables some specific spring test features like autowiring RestTemplate with setted URL

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT

- indicates that test server should be started on random port

- since now you can autowire TestRestTemplate with setted URL to server root

Write hello world test

HelloWorldTest

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

JPA

Connecting with database

JDBC?

```
private static final String SQL_INSERT_SPITTER =
    "insert into spitter (username, password, fullname) values (?, ?, ?)";
private DataSource dataSource;
public void addSpitter(Spitter spitter) {
  Connection conn = null;
  PreparedStatement stmt = null;
  try {
    conn = dataSource.getConnection();
    stmt = conn.prepareStatement(SQL_INSERT_SPITTER);
    stmt.setString(1, spitter.getUsername());
    stmt.setString(2, spitter.getPassword());
    stmt.setString(3, spitter.getFullName());
    stmt.execute();
  } catch (SQLException e) {
    // do something...not sure what, though
  } finally {
    try {
      if (stmt != null) {
        stmt.close();
      }
      if (conn != null) {
        conn.close();
```

WHAT
imgflip.com

Using Spring JDBC Template

```
public void addSpitter(Spitter spitter) {
   jdbcOperations.update(INSERT_SPITTER,
           spitter.getUsername(),
           spitter.getPassword(),
           spitter.getFullName(),
           spitter.getEmail(),
           spitter.isUpdateByEmail());
}
```

meh..

Use Spring Data JPA

```
@Repository
public interface ShoutRepository extends JpaRepository<Shout, Long> {
}
```

Now we can autowire repository component and perform 18 convenient methods for common operations like save, delete, etc...

Using Spring Data mini-DSL

You can add new operations on repository just by adding specifically named methods to interface - Spring magic will implement them at compile time

```
public User findByUsername();
public User findDistinctByEmail();
List<Spitter> readByFirstnameOrLastnameOrderByLastnameAsc(String first, String last);
List<Order> findByCustomerAddressZipCodeOrCustomerNameAndCustomerAddressState();
```

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

Create Shout class

model.Shout

@Entity

- Indicates that annotated class is an entity

@RequestBody

- Indicates that method argument should be bound to request body

Validators:

@Size, @Max, @Min, @NotNull

- validates values assigning to field

@Valid

- performs validation on target (using validators from above)

Jackson serialization/ deserialization

```java
public class ExampleDTO {

   private String desc;

   private int version;

}
```

```json
{
 "desc":"Example data transfer object",
 "version":1
}
```

```json
{
 "desc":"Example data transfer object",
 "version":1
}
```

```java
public class ExampleDTO {

   private String desc;

   private int version;

}
```

```java
public class CarOwner {

   private String name;

   private int age;

   private List<String> cars;
}
```

```json
{
 "name":"Christoph",
 "age":30,
 "cars":
 [
  "Multipla",
  "Polonez",
  "Fiat UNO"
 ]
}
```

Requirements

- No args constructor ({})
- Getters when serializing
- Setters when deserializing

@JsonCreator

In annotated constructor you can set some properties like:

- default values
- required values
- specific names different from field names

```
@JsonCreator
public CarOwner(@JsonProperty(value = "ownerName", required = true, defaultValue = "Anonym")String name,
          @JsonProperty(required = false) int age,
          @JsonProperty(defaultValue = "[\"Fiat 126p\"]") List<String> cars) {
  this.name = name;
  this.age = age;
  this.cars = cars;
}
```

ResponseEntity<P>

ResponseEntity wraps up type P with more elements like response status code, etc.

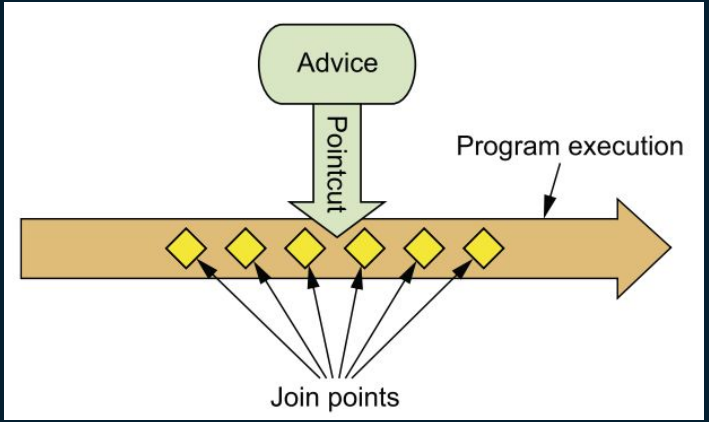
Checkpoints

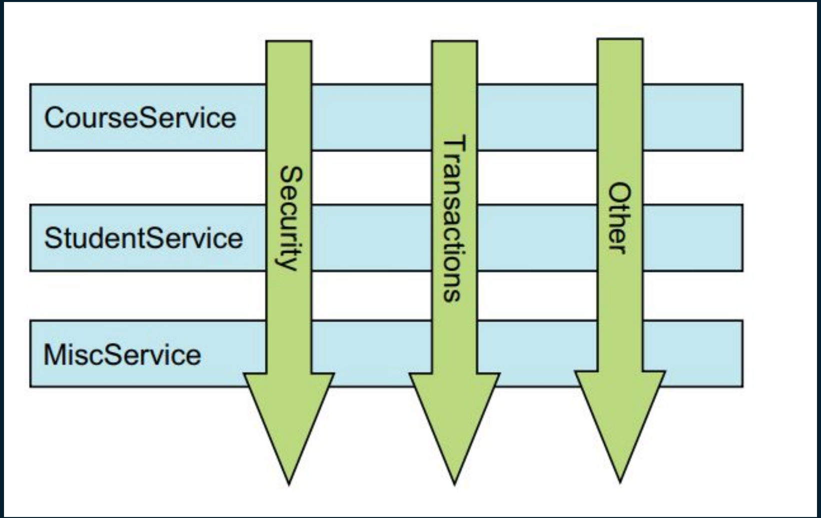
- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

AOP a.k.a Aspect Oriented Programming

In shortcut

- executing logic on a specified event, not dealing with event’s logic. This event could be method being called, exception being thrown, or even a field being modified.


Advice
Pointcut
Program execution
Join points


CourseService
StudentService
MiscService
Security
Transactions
Other

@RestControllerAdvice

@ExceptionHandler(Ex ex) method is being executed every time an Ex is thrown in any Spring Controller

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

Path variables

```
@DeleteMapping("/shouts/{id}")
@ResponseStatus(HttpStatus.OK)
public void deleteShout(@PathVariable long id) {....
```

```
@DeleteMapping("/shouts/{whateva}")
@ResponseStatus(HttpStatus.OK)
public void deleteShout(@PathVariable(name = "whateva") long id) {
```

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

Spring Security

When Spring Security is on classpath...

```
protected void configure(HttpSecurity http) throws Exception {

  http
     .authorizeRequests()
        .anyRequest().authenticated()
        .and()
     .formLogin().and()
     .httpBasic();
}
```

Overriding security config

```
@Override
protected void configure(HttpSecurity http) throws Exception {
   http
           .httpBasic()
           .and()
           .authorizeRequests()
             .antMatchers(HttpMethod.GET, "/allowedForEveryone").permitAll()
             .antMatchers(HttpMethod.POST, "/allowed").permitAll()
             .anyRequest().authenticated()
}
```

UserDetails

Provide UserDetails via UserRepository

Basic HTTP authorization

Just provide header named Authorization:

```
Authorization: Basic aHR0cHdhdGNoOmY=
```

where strange hash is just base64 encoded <username>:<password>

TestRestTemplate withBasicAuth

Duplicates actual instance of TestRestTemplate and appends HTTP Basic Authentication header in every request performed by duplicated TestRestTemplate

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

Accessing authenticated user in controllers

Spring can provide an Authorization object to our authenticated methods

Just extract principals from it and cast to UserDetails implementation type

UriComponentBuilder

Simplifies creating uris with query params

```
String builder = UriComponentsBuilder
        .fromUriString("/shouts/byUser")
        .queryParam("email", randomEmail)
        .toUriString();
```

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

Creating db and db user for application

```
create database spring_boot_workshop;

create user 'appuser'@'localhost' identified by 'ThePassword1';

grant all on spring_boot_workshop.* to 'appuser'@'localhost';
```

Add database access details to application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/spring_boot_workshop
spring.datasource.username=appuser
spring.datasource.password=ThePassword1
```

Different properties for tests

Just create application.properties in test/resources

```
spring.jpa.hibernate.ddl-auto=create
```

create

- creates schema and destroys previous data

create-drop

- creates schema and destroys it at the end of session

update

- updates schema if necessary

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

@Data

Annotating class with @Data adds methods:

- getters for every field
- setters for every non-final field
- constructor for every required field
- equals, hashcode, and toString

Candidates:

Persistences (Shout, User), request and response classes.

Better?

Checkpoints

- Create “Hello, world!” using Spring Boot
- Enable adding and reading shouts
- Add error handling
- Add new fields to shout
- Add user to project
- Enable Spring Security
- Add user’s features to project
- Adding MySQL to project
- Improving (?) with Lombok

END