Internet Application Design and Implementation – 2019/2020

MidTerm Test November 2, 2020

Notes: The test is closed book and has a duration of 1h30m. You may bring 2 handwriten A4 pages. There are 10 multiple choice answer questions and 3 open answer questions that should be answered in the provided answer sheets. Correct answers in multiple choice questions add 1 point, the first 3 wrong answers have a penalty of 1/8 of the value of a correct answer, the following wrong answers have a penalty of 1/4 of the value of a correct answer. Open answers are awarded 3, 4 and 3 points respectively. Do not unstaple the questions. You may use a pencil.

Version: A

Name:____Number:____________

Q-1 Consider the following request using the command line tool httpie:

```
http GET :8080/trips/{tripId}/passengers?startDate="2020-12-20"&endDate="2020-12-31"
```

Select the declaration that correctly implements this request in Spring using kotlin.

A -
```
@RestController @RequestMapping("/trips/{tripId}") class TripController {
  ...
  @GetMapping("/passengers")
  fun getPassengers(@PathVariable tripId:Long ,
                    @RequestParam startDate:Date,
                    @RequestParam endDate:Date ) : List<Trip> {...}
  ...
```

B -
```
@RestController @RequestMapping("/trips") class TripController {
  ...
  @GetMapping("/{tripId}/passengers?startdate={startdate}&enddate={enddate}")
  fun getPassengers(@PathVariable tripId:Long,
                    @PathVariable startDate:Date,
                    @PathVariable endDate:Date ) : List<Trip> {...}
  ...
```

C -
```
@RestController @RequestMapping("/trips") class TripController {
  ...
  @GetMapping("/{tripId}/passengers")
  fun getPassengers(@PathVariable tripId:Long ,
        @PathVariable startDate:Date,
        @PathVariable endDate:Date ) : List<Trip> {...}
  ...
```

D -
```
@RestController @RequestMapping("/trips/{tripId}") class TripController {
  ...
  @GetMapping("/passengers")
  fun getPassengers(@PathVariable tripId:Long ,
                    @RequestBody startDate:Date,
                    @RequestBody endDate:Date ) : List<Trip> {...}
  ...
```

Q-2 Consider the following JPQL query "`select o from Order o join fetch o.products p where o.year = :year`"
Select the **CORRECT** sentence below:

A - The query above triggers another query to load the objects contained in the relation defined by field `products`.

B - The query above loads into memory all the objects of class `Order` and the objects in the collection `products` of class `Order` for the year given as parameter.

C - The relation between classes Order and Product is marked with fetch type `Eager`.

D - Any iteration of a collection `products` in an object of class `Order` given by the query above, should be performed in the same transaction as the query.

Q-3 Consider the following pair request/response and evaluate its maturity level according to the Richardson maturity scale.

Request 1:
```
http GET /api/movies/2/cast
```
Response 1:
```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8

[{"birthyear": 1981, "name": "Elijah Wood"}, {"birthyear": 1939, "name": "Ian McKellen"}]
```
Request 2:
```
http PUT /api/movies/2 title="The Lord of the Rings" director="Peter Jackson"
```
Response 2:
```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8

{ "title": "The Lord of the Rings", "director": "Peter Jackson" }
```

A - Level 3: Hypertext as the engine of applications state (HATEOAS)

B - Level 2: interactions with URI resources using different http verbs

C - Level 0: Plain old XML (POX)

D - Level 1: Multiple URI based resources and single verbs

Q-4 One wants to test the service that provides data about a hotel room. Consider the skeleton of the test below where `rooms` is a reference to a repository for the data class `Room`. Select the fragment that correctly completes the test (without mocking).

```
@Test @WithMockUser(username = "user1", roles = "REGISTERED")
fun ‘testing the retrieval of one Room‘() {
  val roomId = rooms.searchByName(BIG_SUITE_NAME)[0].id
  mockMvc.perform(get("/rooms/" + roomId ))
  ...
}
```

A -
```
.andExpect(status().isOk)
.content("{\"id\":\"0\", \"name\":\"Big Suite\"}")
.andReturn()
```

B -
```
.andExpect(status().isOk)
.andExpect(content().contentType(MediaType.APPLICATION_JSON_UTF8))
.andExpect(jsonPath("$[0].name", is(BIG_SUITE_NAME)))
.andReturn()
```

C -
```
.andExpect(status().isOk
.andExpect(content().contentType(MediaType.APPLICATION_JSON_UTF8))
.andExpect(content("name", equalsTo(BIG_SUITE_NAME)))
.andReturn()
```

D -
```
.andExpect(status().isOk)
.andExpect(content().contentType(MediaType.APPLICATION_JSON_UTF8))
.andExpect(jsonPath("$.name", is(BIG_SUITE_NAME)))
.andReturn()
```

Q-5 Consider the following JPA Entities.

```
@Entity
data class Veterinarian(@Id val id:Long, val name:String, @OneToMany val appointments:List<Appointment>)

@Entity
data class Appointment(@Id val id:Long, val startDate:Date, val endDate: Date,
                       @ManyToOne val vet:Veterinarian, @ManyToOne val pet:Pet )

@Entity
data class Pet (@Id val id:Long, val name:String, val species:String)
```

Select the most efficient way to implement the query associated to the following request, here depicted using the `httpie` command line syntax:

```
http GET :8080/veterenerians/{id}/appointments?startdate="2019-12-20"&enddate="2019-12-31"
```

A -
```
@Query("select a from Appointments a join Vets v where v.id = :id "+
       "and a.startDate between :startDate and :endDate")
fun findByVetAndDate(id:Long, startDate:Date, endDate:Date):List<Appointment>
```

B -
```
@Query("select a from Veterinarian v join v.appointments a "+
        "where v.id = :id and a.startDate = :startDate")
fun findByVetAndDate(id:Long, startDate:Date, endDate:Date):List<Appointment>
```

C -
```
@Query("select a from Veterinarian v join v.appointments a where v.id = :id "+
        "and a.startDate between :startDate and :endDate")
fun findByVetAndDate(id:Long, startDate:Date, endDate:Date):List<Appointment>
```

D -
```
fun findByVetAndDate(id:Long, startDate:Date, endDate:Date):List<Appointment>
```

Q-6 A passenger from a trip may access to the details of their trips in an air line. A security policy should prevent other passengers from accessing their data. Consider the following Kotlin code to represent a controller and corresponding annotations.

```
@GetMapping("/passengers/{id}/trips/{tid}")
@CanViewTrip fun getPassengersTrip(@PathVariable long id, @PathVariable long tid) {...}
```

Select the appropriate Kotlin implementation for the security policy.

A -
```
@PreAuthorize("hasRole('ADMIN') or hasRole('GUEST')")
@interface CanViewTrip {}
```

B -
```
@PreAuthorize("hasRole('ADMIN') or @securityService.ownerOfTrip(#id, #tid)")
@interface CanViewTrip {}
```

C -
```
@PreAuthorize("hasRole('ADMIN') or @securityService.ownerOfTrip(principal, #id, #tid)")
@interface CanViewTrip {}
```

D -
```
@PreAuthorize("hasRole('ADMIN') or @securityService.isPrincipal(principal, #id)")
@interface CanViewTrip {}
```

Q-7 Spring is a component-based framework that allows the assembling of applications by composing different beans together. **Select the correct method to assemble components.**

A - Dependency injection via the declaration of lazily initialized fields and the `Autowired` annotation.

B - Assembly is performed by configuration files written in XML.

C - Assembly is performed by explicitly instantiating objects and linking components with object references.

D - Dependency injection in the parameters of the constructors of the components or via the `Autowired` annotation.

Q-8 The JPA specification denotes an underlying database schema. **Select the correct sentence.**

A - The database schema is always adjusted to the JPA schema modifications whenever the application starts.

B - A `@ManyToMany` annotation can be instantiated without an auxiliary table.

C - A `@ManyToOne` annotation with no extra parameters corresponds to an extra table in the database to implement the relation.

D - A `@ManyToOne` annotation is always associated to a collection of objects in the class where it is used.

Q-9 This question is about the use of frameworks in the development of Internet applications. **Select the correct option**.

A - Frameworks like "Ruby on Rails" are domain specific languages that are used to implement web applications.

B - Spring is a component-based framework whose main composition and extension mechanism is class inheritance.

C - Software frameworks provide a safer and more productive way by implementing design and architectural patterns.

D - SpringBoot is a component based framework that implements an instance of the MVC pattern.

Q-10 In a SpringBoot application, consider the following annotations in a controller class

```
@RestController @RequestMapping(value="/departments") class DepartmentsController { ... }
```

Select the correct sentence from the options below.

A - The methods in this class with a `RequestMapping` annotation will serve requests at the URL specified on each method and their responses will be automatically translated to JSON.

B - The methods in this class with a `RequestMapping` annotation will append a sufix to the URL `/departments` and translate their requests and responses automatically from, and to, JSON.

C - All methods that serve REST requests have return type `ResponseEntity` and explicitly translate objects to JSON.

D - All methods starting with `/departments` will be redirected to the methods of this class and their responses will be automatically translated to JSON.

Part 2

System Description: Consider a system for the management of repairs on a car repair shop. The users of the system are either clients or employees of the car repair shop. When the car arrives at the shop a repair request is submitted in the system, and the client can edit their requests. A request for a repair consists of a description, a car identifier, and a date. The statuses of a repair request can be ”created”, ”submitted”, ”finished”, and ”paid”. Clients can always see their repair requests, but can only edit their repair requests while the status has value ”created”. Employees of the repair shop can only finish a repair request when they have the status ”submitted”. The main resource here is a repair request.

The system described above is the development context for all questions below.

Q-11 [4 pts] Define a RESTful interface, Level 2 in the Richardson maturity scale, that defines the operations on the available expense reports to both employees and heads of departments. List all the endpoints by means of a Kotlin interface using data classes for DTO objects. No swagger annotations are needed.

Q-12 [3 pts] Define the JPA (data) classes for the system described above and relate them correctly using JPA annotations.

Q-13 [3 pts] Define two security policies, annotations and corresponding services, that regulates the access for reading, and also for changing the status of the main resource of the scenario above.