Lecture 8 - IADI 18 Nov 2021

2.

```
mockMVC.perform(get("/rooms/" + roomId))
.andExpect(status.isOK())
.andExpect(content().string("{roomId:1, description:{...
.andExpect(jsonPath("$.name", is(BIG_SUITE_Name
```

→ assertEquals

name: "Bigsu

$ → the root of the result

$.name

[ ] jsonPath($, is("Bla"))

($[0], is(...))

= {-}

3.

```
data class HotelVisit {
    @Id @GeneratedValue var id: Long,
    var name: String,
    @ManyToMany var rooms: List<Room>,
    var startDate: Date
    var endDate: Date }
}
```

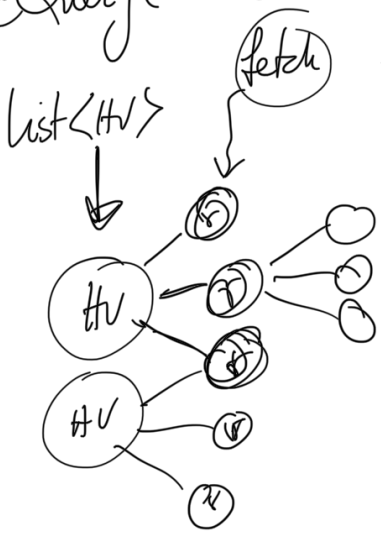
```
@Query("select v from HotelVisits inner join fetch
    v.rooms where v.startDate between :start
    and :end or v.endDate between :start
```

and : end ")

fun findVisitsBetweenStartDateAndEndDate (start : Date,
end : Date)
: List<HotelVisit

X

@Query (" select r from HotelVisits v
inner join ~~fetch~~ v.rooms r
where .... ") : List<Room>



Part 2

Q5

```
@Entity
data class ExpenseReportDAO (
    @Id @GeneratedValue id: Long,
    description : String,
    date : Date,
    amount : Int
    state : ERState
@ManyToOne employee : Employee )

data class Employee (
    @Id @GeneratedValue id: Long,
    name: string,
@ManyToOne department : Department,
@OneToMany expenseReports : List<ExpenseReport> )
```

```
enum ERState
{ Created,
  Submitted,
  Approved,
  Paid
}
```

data class Department(
@Id @GeneratedValue id: Long,
name: String,
@OneToOne @ManyToOne headOfDep: Employee
@OneToMany employees: list<Employee>
)

@4

@RequestMapping("/reports")
interface ExpenseReportAPI : GenAPI<...> ?

@GetMapping("")
fun getAll(): list<ExpenseReportsListDTO>

? @PostMapping("") fun addOne(

1. GET /reports ← @GetMapping("/reports") fun getAllReports(): list<ReportListDT

@GetMapping("/reports/{id}") fun getOne(id: long): ReportLong Optional<

2. GET /reports/id

3. PUT /reports/id (ex: Approve, Edit)

DELETE /report...

4. GET /employees/id/reports ← reports of empl

5. POST /employees/id/reports ← new Report for employee

```
@PostMapping("/employees/{id}/reports
fun addOne(@PathVariable id,
           @RequestBody report: Report
```

: Vo

: Un

6. GET /departments/id/reports?state = "submitted" or "approved" or "paid"

or

GET /departments/id/submittedReports

→ val id: Long

data class ReportListDTO(val description: String
val date: Date
val amount: Int,
→ val state: ERState,
→ val employee_id: Long)

link to employee (String? Object?

data class ReportDTO(...) // no id // no employee // no state

data class ReportLongDTO(...) ← full employee information + just the name

---

Q6

1. Heads of department can only approve ER in the state "submitted" and from their employees

```
@PreAuthorize( CanEditER.condition )
@interface CanEditExpenseReport {
    object companion {
        val condition = "
            @mySecurityService.OwnER( id, prin
            or
            @.....IcanApproveER( id, princip
                                        report)
    }
}
```

```
@PutMapping("/reports/{id}")
fun editOne(@PathVariable id:Long,
            @RequestBody report:ReportDTO):Void
```

```
@Service
class SecurityService(reports: RepoReports)

fun OwnER(id:Long, principal: Principal):Boole
=
    reports.findById(id)
        .orElseReturn(false)
        .let { it: ReportDAO
            it.employee.username == principal
                                    .usernam
            &&
            it.state == ERState.created
        }

fun ICanApproveER(id:Long
```

quer ... (a ...)   principal : Principal &
                                  report : ReportDTO))

```
= reports.findById(id)
      .orElseReturn(false)
      .let { // it: ReportDAO  <-
          it.state == ERState.submitted
          &&
          it.employee.department.headOfDep.
                                       username
            == principal.username
          &&
            report.amount == it.amount
          && report.desc == it.description
          && report.date == it.date
          && report.state == ERState.approv
      }
```

2. Employees can only see their ERs.