Modern Backend Systems

Don’t end up with a beast…

Georgios Voulgarakis

CERN

What you need

- **ssh** client with **X11 forwarding**
  - Windows: Xming X Server: https://sourceforge.net/projects/xming/
  - Mac: Xquartz https://www.xquartz.org/

Or:

- **Eclipse** with **Maven**

Exercises and presentation material is available at

Specifications

- **Applicants post their “job traits”**
- **Vacancies are posted with their required “job traits”**
- **Applications have a “fit” criteria, of the Applicant to the job post, deriving from matching “job traits”**
- **HR should be notified about “good matching applications”**
- **Backend System entities should be persisted in a DB**

Entities

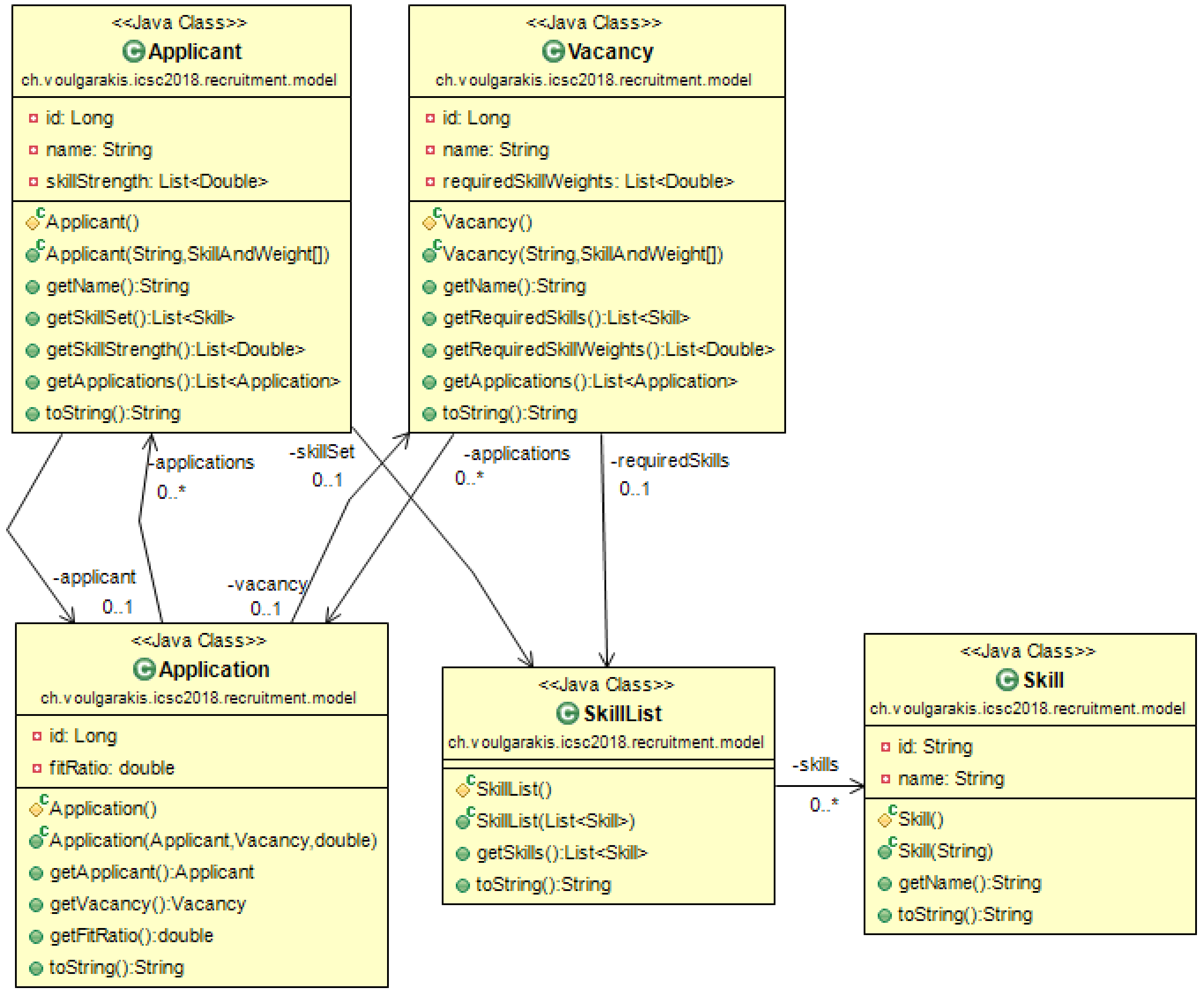
First ideas…:

- **Job traits…?**
- **Applicant**
- **Vacancy**
- **Application, fit?**

On a second thought…:

- **Store Entities in DB.** → Data Access Object -Repository
- **DB relationships?** → Applicant, Vacancy, Application, Skill Cascade on DB? Or handled by Service?
- **DB Id?** → When should it be assigned? By high-level language, or DB?

Entities



<<Java Class>>
Applicant
ch.voulgarakis.icsc2018.recruitment.model
id: Long
name: String
skillStrength: List<Double>
Applicant()
Applicant(String,SkillAndWeight[])
getName():String
getSkillSet():List<Skill>
getSkillStrength():List<Double>
getApplications():List<Application>
toString():String
<<Java Class>>
Vacancy
ch.voulgarakis.icsc2018.recruitment.model
id: Long
name: String
requiredSkillWeights: List<Double>
Vacancy()
Vacancy(String,SkillAndWeight[])
getName():String
getRequiredSkills():List<Skill>
getRequiredSkillWeights():List<Double>
getApplications():List<Application>
toString():String
-applications 0..*
-skillSet 0..1
-applications 0..*
-requiredSkills 0..1
-applicant 0..1
-vacancy 0..1
<<Java Class>>
Application
ch.voulgarakis.icsc2018.recruitment.model
id: Long
fitRatio: double
Application()
Application(Applicant,Vacancy,double)
getApplicant():Applicant
getVacancy():Vacancy
getFitRatio():double
toString():String
<<Java Class>>
SkillList
ch.voulgarakis.icsc2018.recruitment.model
SkillList()
SkillList(List<Skill>)
getSkills():List<Skill>
toString():String
-skills 0..*
<<Java Class>>
Skill
ch.voulgarakis.icsc2018.recruitment.model
id: String
name: String
Skill()
Skill(String)
getName():String
toString():String

Exercise 1: JPA - ORM

In *SpringBoot* project, open:
`ch.voulgarakis.recruitment.tests.TestApplicant;`
You will find the exercise instructions there.

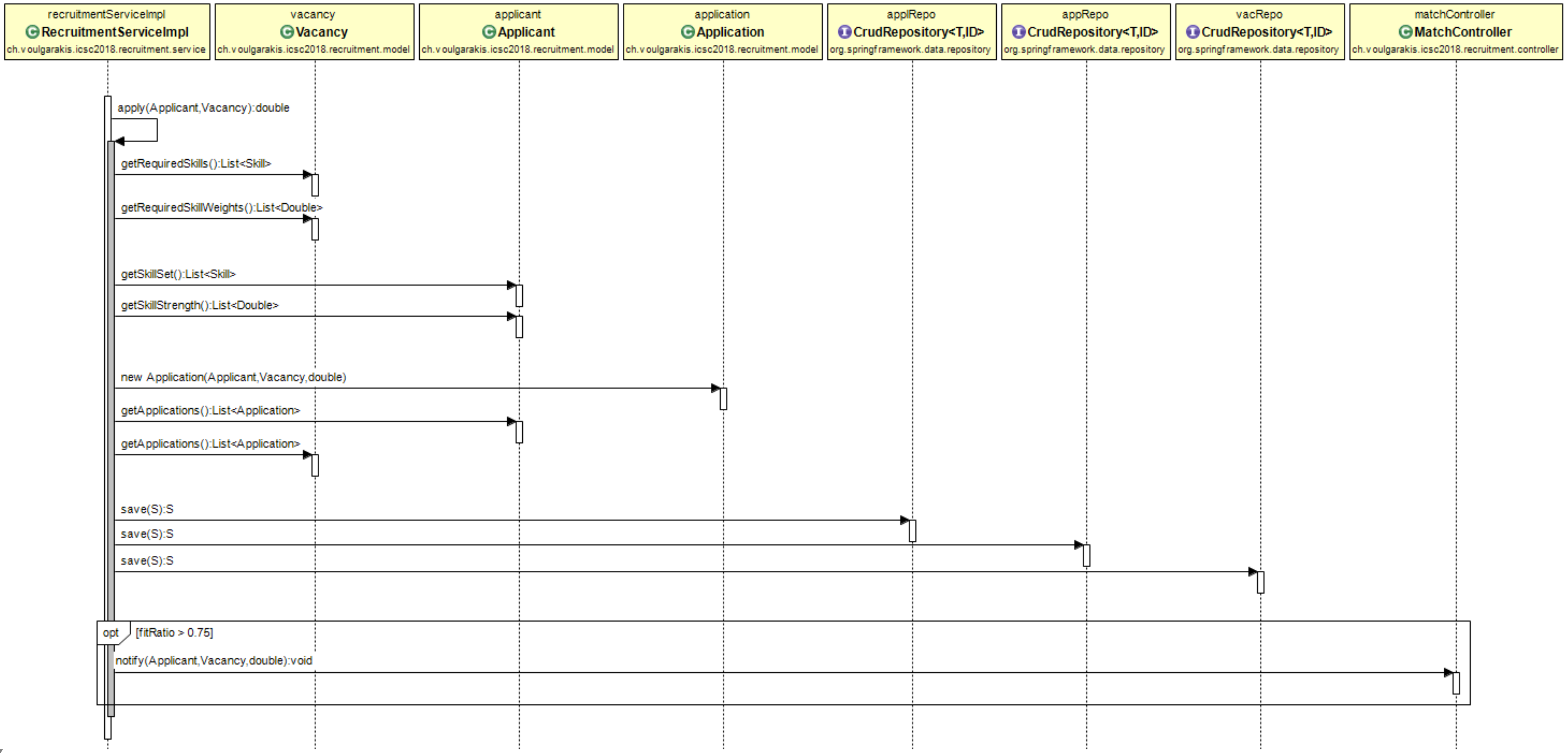
Hint:

- @Entity
- @Id @GeneratedValue
- @ManyToMany @OneToMany @ManyToOne
- You will need a no-args protected constructor for the JSON conversion to work

Service

Hint: **@Autowired** for bean injection

- **Load/Save Applicant/Vacancy**
- **Apply & Notify HR about applications with good fit? → websockets?**


recruitmentServiceImpl
RecruitmentServiceImpl
ch.voulgarakis.icsc2018.recruitment.service
vacancy
Vacancy
ch.voulgarakis.icsc2018.recruitment.model
applicant
Applicant
ch.voulgarakis.icsc2018.recruitment.model
application
Application
ch.voulgarakis.icsc2018.recruitment.model
applRepo
CrudRepository<T,ID>
org.springframework.data.repository
appRepo
CrudRepository<T,ID>
org.springframework.data.repository
vacRepo
CrudRepository<T,ID>
org.springframework.data.repository
matchController
MatchController
ch.voulgarakis.icsc2018.recruitment.controller
apply(Applicant,Vacancy):double
getRequiredSkills():List<Skill>
getRequiredSkillWeights():List<Double>
getSkillSet():List<Skill>
getSkillStrength():List<Double>
new Application(Applicant,Vacancy,double)
getApplications():List<Application>
getApplications():List<Application>
save(S):S
save(S):S
save(S):S
opt
[fitRatio > 0.75]
notify(Applicant,Vacancy,double):void

Exercise 2: Service implementation

In *SpringBoot* project, open:
`ch.voulgarakis.recruitment.tests.TestService;`
You will find the exercise instructions there.

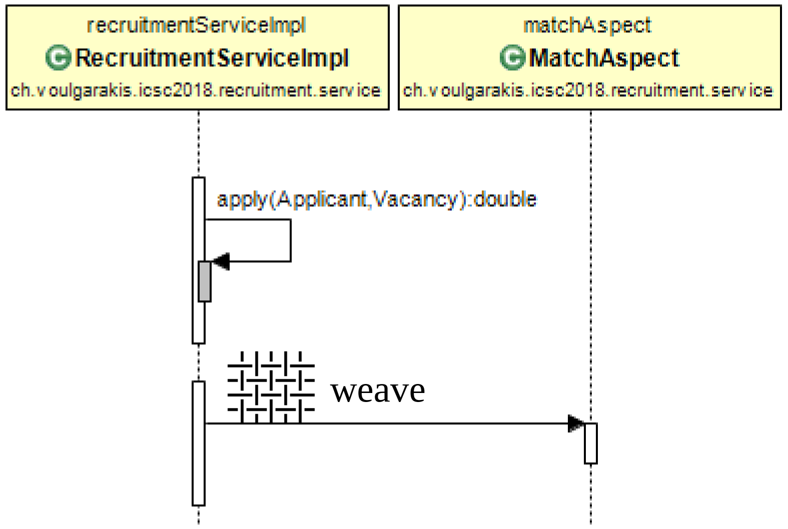
Hint:

- @Autowired -> Injection (for the Application Repository)
- applicationRepository.save() -> to persist/save/update an entity

Cross-Cutting Concerns

- **Consider the “notification of the HR” as a Cross-Cutting concern?**
  → Aspect?

- **Capture apply(..) method invocation, and after returning the “fitRatio”, check if notification to HR should be sent.**


recruitmentServiceImpl
RecruitmentServiceImpl
ch.voulgarakis.icsc2018.recruitment.service
matchAspect
MatchAspect
ch.voulgarakis.icsc2018.recruitment.service
apply(Applicant,Vacancy):double
weave

Exercise 3: Cross Cutting Concerns

In *SpringBoot* project, open:
`ch.voulgarakis.recruitment.tests.TestService;`
You will find the exercise instructions there.

Hint:

- @AfterReturning,
  → Capture the successful (no exception thrown) return of a method signature
- **`execution(int calculate(..)) && args(applicant,vacancy)`**
  → method signature: execution of method return *int* type, with name *calculate*, and taking arguments *applicant & vacancy*

JPA - ORM

- What happens if exception is throw when working on the service layer? Ie: saveApplicant() fails?
- **Transaction: propagation? Isolation?**

☐ REQUIRED

- ➢ support a current transaction, create a new one if none exists

```
@Transactional(propagation = Propagation.REQUIRED)
public Long process(Account account, Order order){}
```

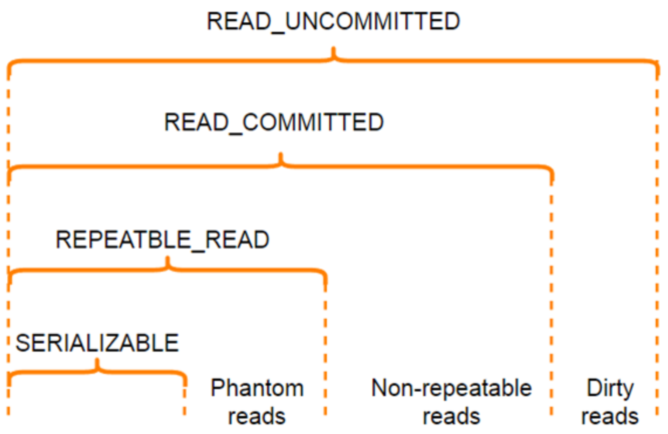
☐ REQUIRED_NEW

- ➢ create a new transaction, suspend the current if one exists

```
@Transactional(propagation = Propagation.REQUIRED_NEW)
public Long process(Account account, Order order){}
```

☐ NESTED

- ➢ single physical transaction with multiple savepoints



Exercise 4: Transaction

In *SpringBoot* project, open:
`ch.voulgarakis.recruitment.tests.TestTransactional;`
You will find the exercise instructions there.

Hint:

- Familiar with the concept of Transaction?
- You can make use of the transaction manager…
- Or check out the @Transactional annotation…

REST

Our Recruitment Service REST endpoints:

- GET /recruitment/applicant/{name}
- POST /recruitment/applicant
- GET /recruitment/vacancy/{name}
- POST /recruitment/vacancy
- PUT /recruitment/apply?applicantId=3&vacancyId=10
- PUT /recruitment/apply?applicantName=Claus&vacancyId=Santa

Exercise 5: REST

In *SpringBoot* project, open:
`ch.voulgarakis.recruitment.tests.TestREST;`
You will find the exercise instructions there.

Hint:

- Look at the existing implementation of the endpoint:
  PUT /recruitment/apply?applicantId=3&vacancyId=10
- Look at the implementation of ***loadApplicant()*** & ***loadVacancy()***
- @PathParam, @PathVariable
- ResponseEntity
  → Response with Http Status Code & Body supporting JSON/XML conversion

Exercise 6: Reactive Streams

In *SpringBoot* project, open:
`ch.voulgarakis.recruitment.tests.TestRX;`
You will find the exercise instructions there.

Hint:

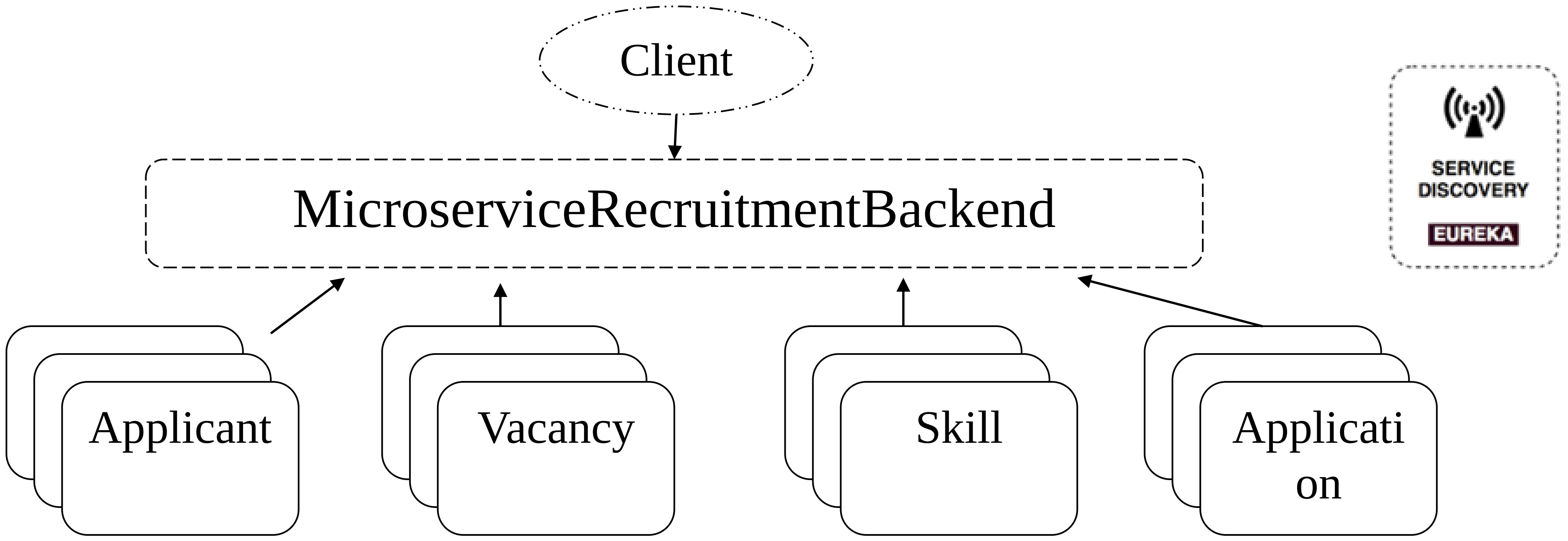
- take(),
- takeLast()
- map()
- buffer()

MicroServices

- Break our backend into 5 Microservices
- These support **CRUD** operations.

Skill, Application, Vacancy, Applicant → CRUD

- Use **Ribbon, Eureka, Feign** to redesign RecruitmentService, and **Load Balance** between multiple instances of these 4 types of microservices.



Exercise 7: Microservices

In *SpringBootMicroservices* project, open:
`ch.voulgarakis.recruitment.tests.TestMicroservices;`
You will find the exercise instructions there.

Hint:

- rest.postForEntity
- http://service-name/....
- httpResponse.getStatusCode()
- httpResponse.getBody()

Github

- **https://github.com/gevoulga/spring-boot**
- **https://github.com/gevoulga/microservices**