Domain Driven Design

Mikael Sundberg
Pingpong
@msundb

Why do we use DDD?

- Have a clear language.
- To highlight uncertainties.
- Challenge ourselves.

Example

What makes a Course?

- Material to learn
- Questions to verify you understand the material
- Students

Example

What makes a Course?

- Material to learn.
- ~~Questions to verify you understand the material~~ - Maybe, the evaluation probably not.
- ~~Students~~ - No, lets call it a Participation.

Can we Rename Course? Can we be more precise?

Pågående

Butikssäljare på …

Som butikssäljare på Teknikm…

Fotboll utan fylla

Alkoholprevension på idrottsa…

Internkontroll för …

Internkontroll och styrning är …

Name and description

Progress

Alla kurser och program

Odla i pallkrage

Lär dig smarta sätt att få stor skör…

STARTA KURSEN

Basic Principles of …

Baloo Learning offers an introducti…

STARTA KURSEN

Grunderna i GDPR

Baloo Learning bjuder på en kurs i …

STARTA KURSEN

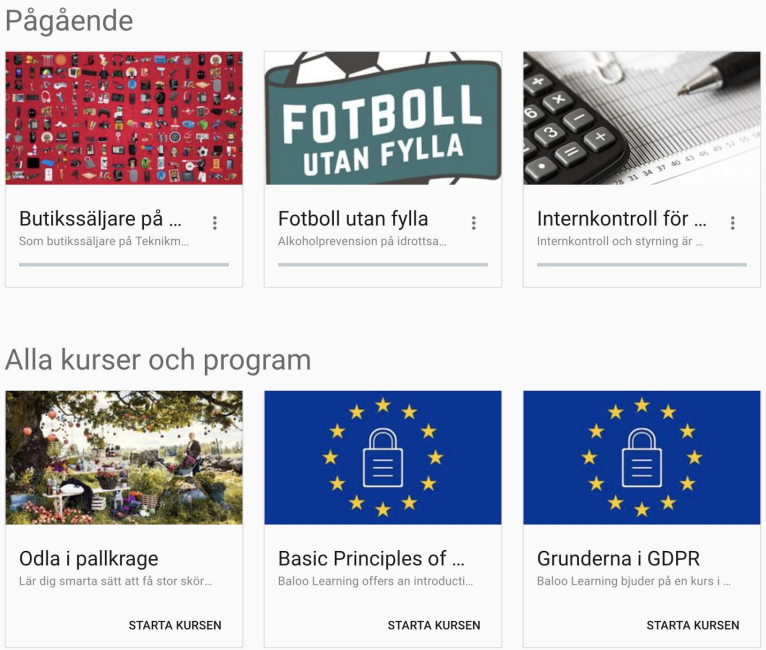
Is the course started?

Course catalog

- Course participation
- Course progress
- Course information
- etc...

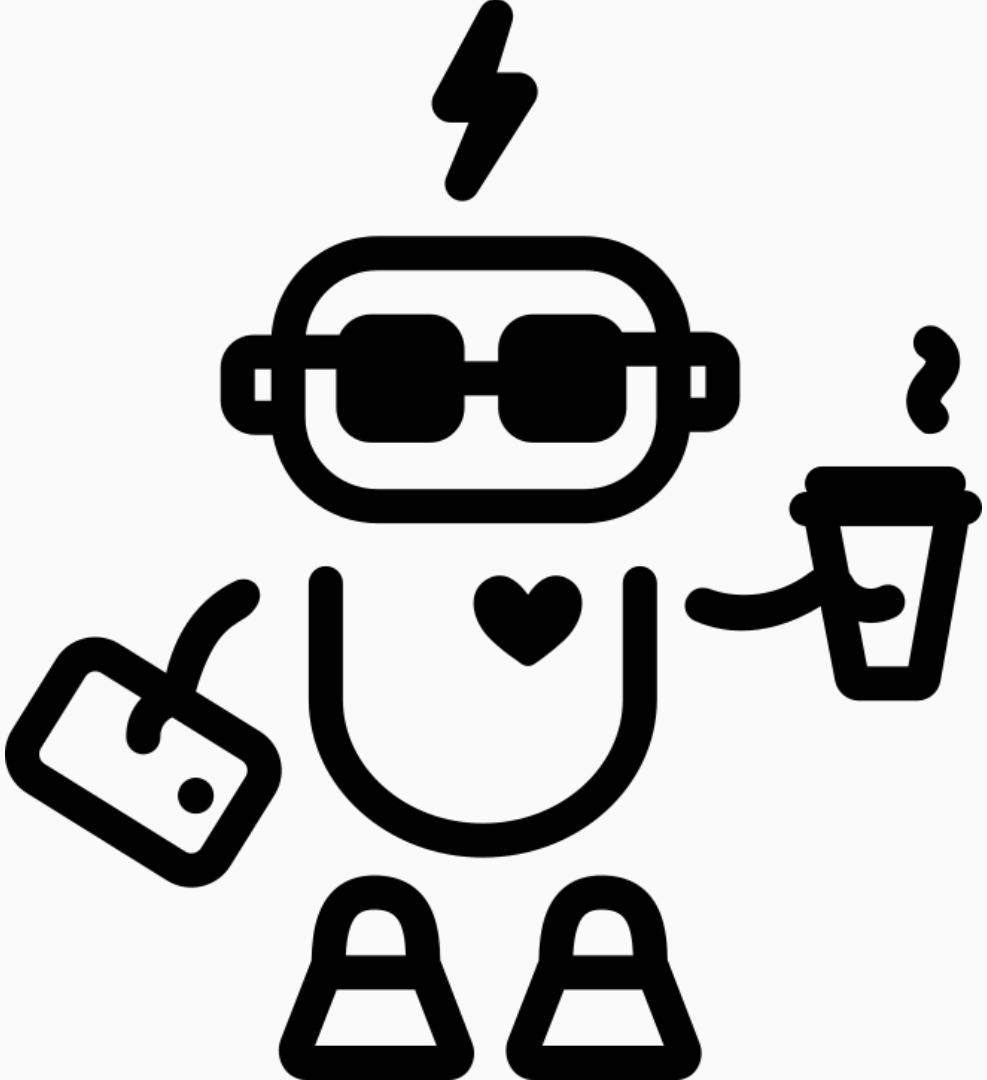
That sounds like many domains?

What domain is this? And in what context?



Baloo

One app/website
many domains



Welcome to Baloo Learning


Create


Learn


Analyse


Achievements


Plan

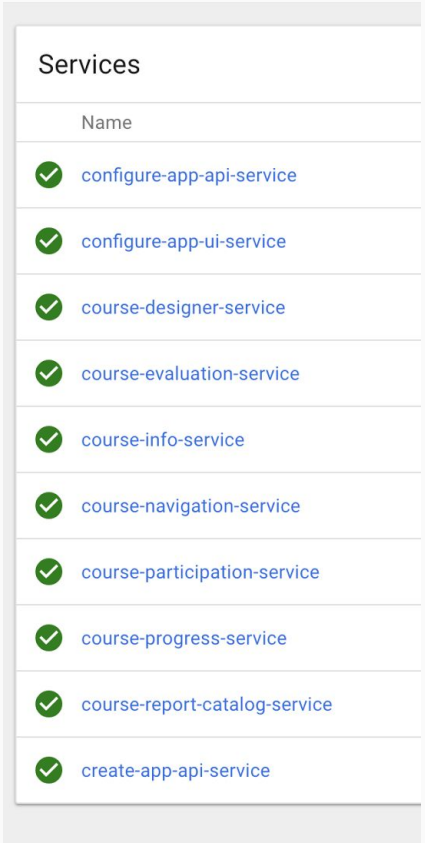
Admin

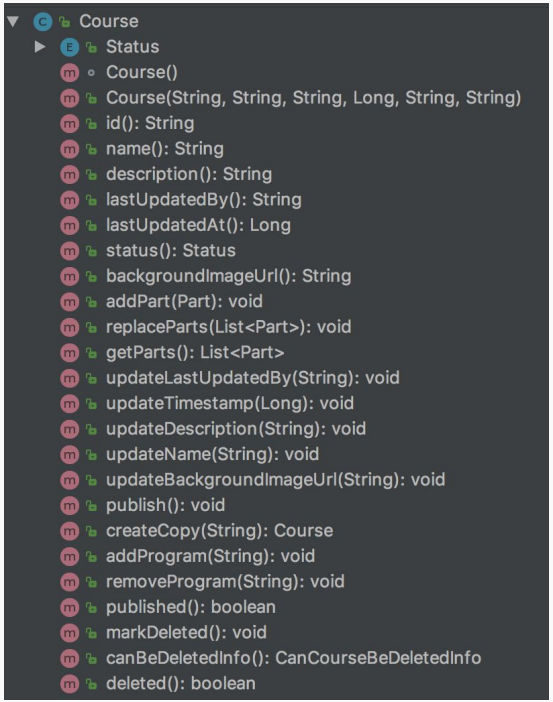

App Access

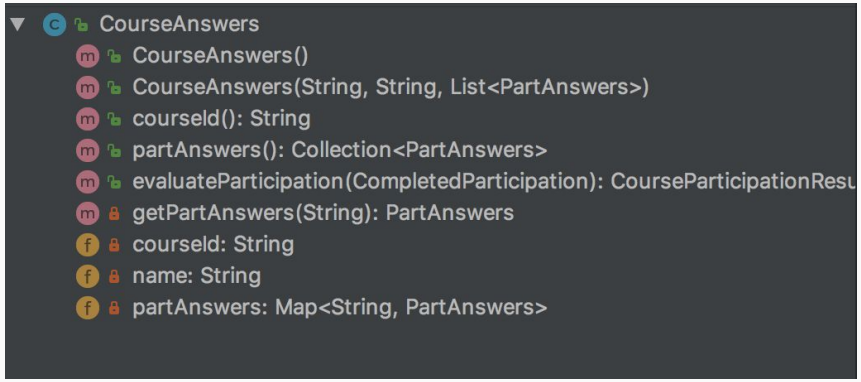

Users & Groups


Configure

Each domain/context is a service. (not all services is a domain or context though)



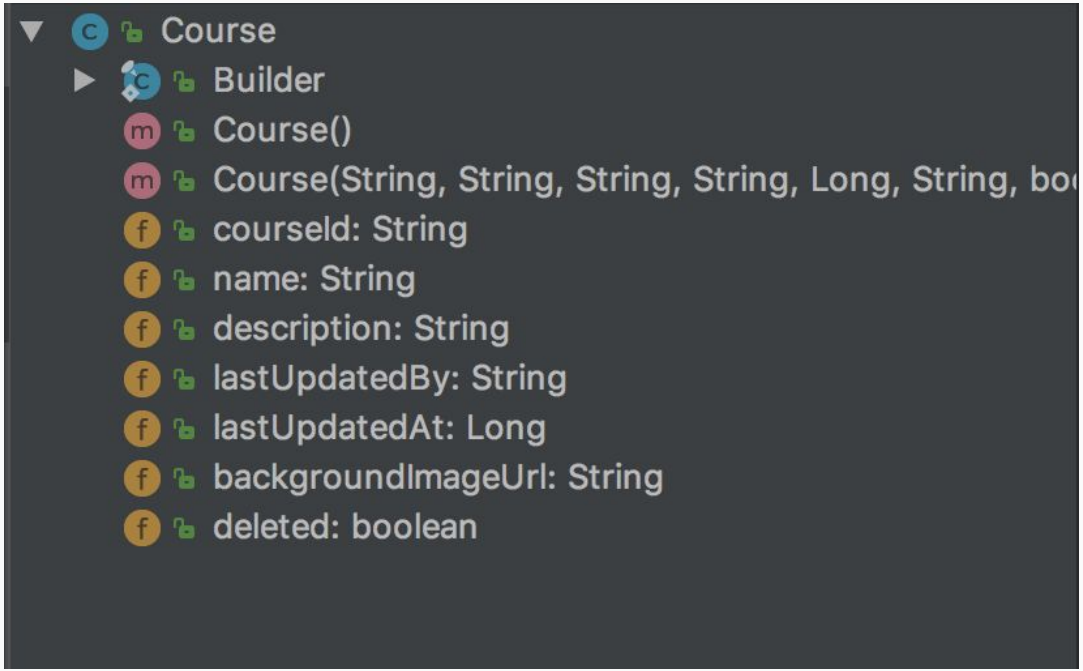




Many different Courses

Domain objects or data bags

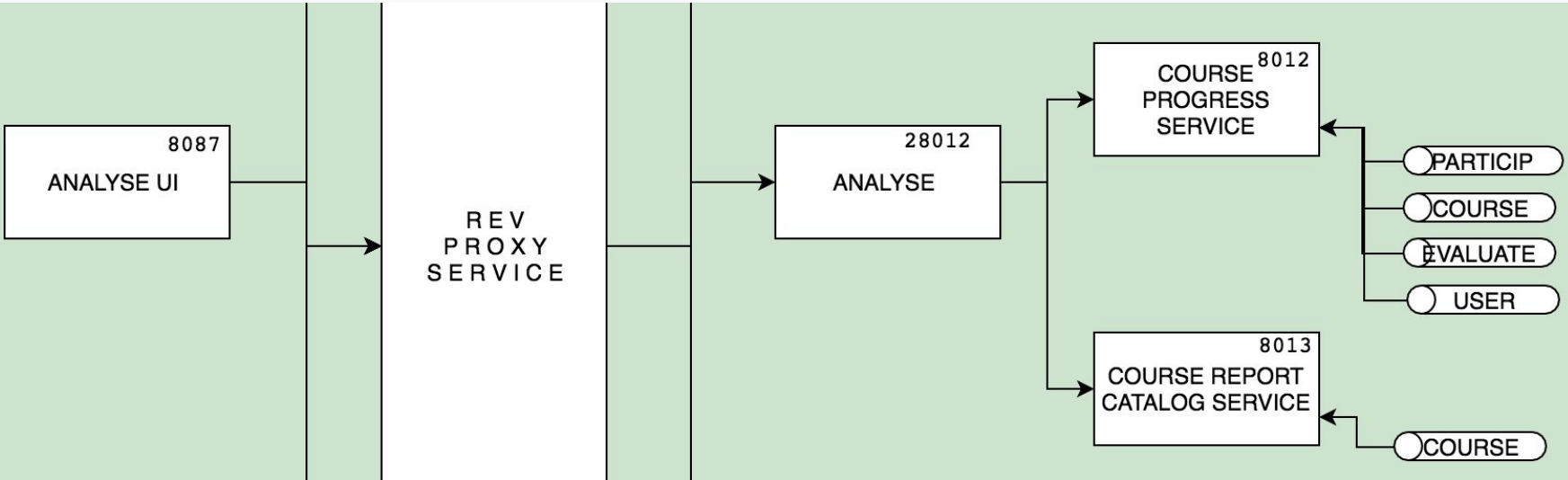
Not everything is a domain object.
Sometimes you just have data.



Some things we live by

Keep your own state

A service should not talk to other services. Listen to messages and keep your own state. Your Course is different from your neighbours course.



Creating a new service

Bootstrap = do http call on spin up to fetch missing data

```
private fun spinUpService(programRepositoryFactory: RepositoryFactory<ProgramRepository>,
    val client = org.glassfish.jersey.client.JerseyClientBuilder.createClient()
    logger.info("Looking for customers")
    val response = client.target("http://customer-config-service/").path("customers").req
    if (response.statusInfo.family != Response.Status.Family.SUCCESSFUL) {
        throw RuntimeException("http call error")
    }
    val customers: List<String> = response.readEntity(List::class.java) as List<String>
    logger.info("Found [{}] customers", customers.size)

    customers.forEach { customer ->
        programRepositoryFactory.get(CustomerId(customer)).drop()
        val courseInfoResponse = client.target("http://program-info-service/").path("cust
        if (courseInfoResponse.statusInfo.family != Response.Status.Family.SUCCESSFUL) {
            throw RuntimeException("http call error")
        }
        val programsDto: ProgramsDto = objectMapper.readValue(courseInfoResponse.readEnti
        logger.info("Found [{}] programs", programsDto.programs.size)
        programsDto.programs.forEach { program ->
            logger.info("Saving program [{}] in db", program.programId)
            programRepositoryFactory.get(CustomerId(customer)).saveOrUpdate(Program(progr
        }
    }
}
```

Revert "CORP-499 Spin up non deleted programs"

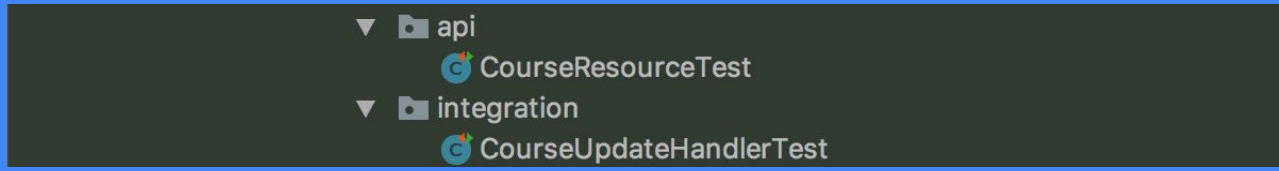
CORP-499 Spin up non deleted programs

Be extremely specific

Not being specific enough can cause your domains/bounded contexts to grow more than you want. And adds confusion.

If you add more info in service A, because downstream service B needs it, something could be fishy.

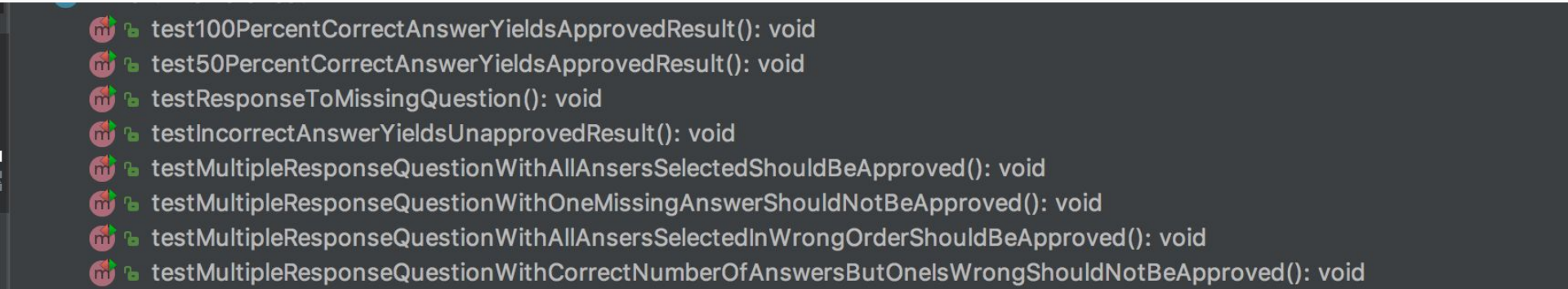
Testing



0 tests using more than 1 service

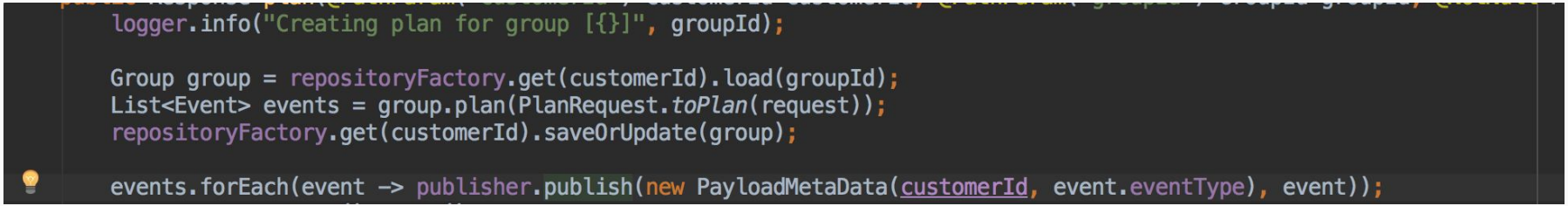
Heavy focus on unit testing domain objects

Most logic should be in domain objects, and they are easy to test!



Domain models can emit events

The domain object knows what happend.

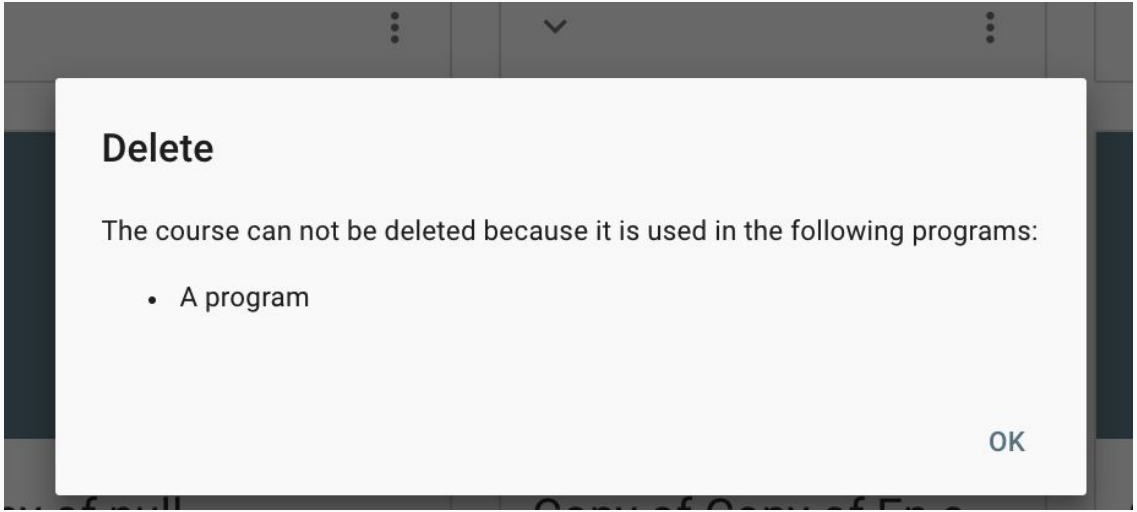


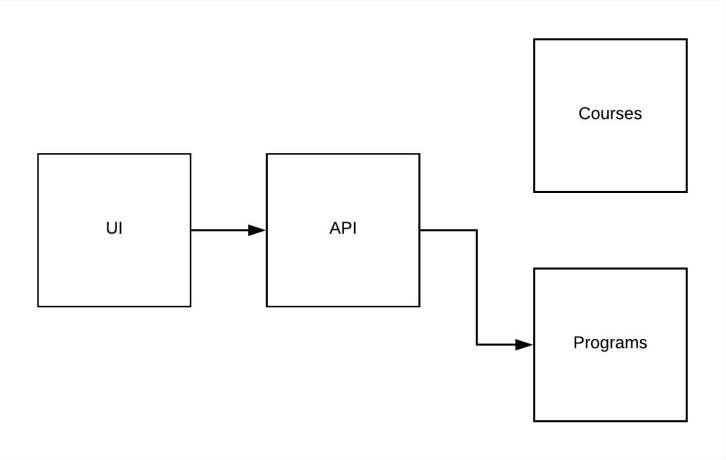
Examples

Can a course be deleted?

Business rule: A course can be deleted, unless its is in a program.

We could ask program-service if it has any program with this course.



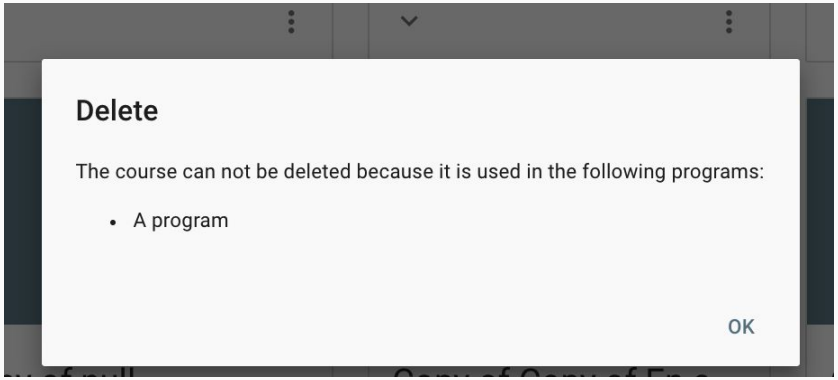


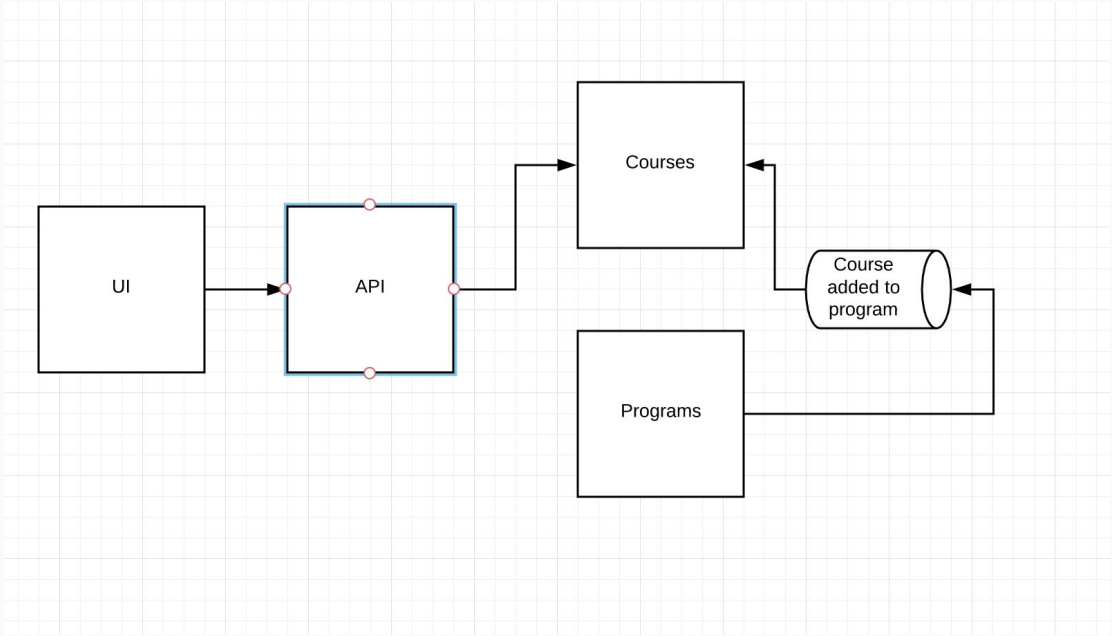
Can a course be deleted?

Business rule: A course can be deleted, unless its is in a program.

We could ask program-service if it has any program with this course.

But, program service should not be responsible for if a course can be deleted.





Can a course be deleted?

```
@GET
@Path("{courseId}/can-be-deleted")
public Response canCourseBeDeleted(@PathParam("courseId") String courseId) {
    retrofit2.Response<CanCourseBeDeletedDto> canCourseBeDeletedDtoResponse = courseDesignerService.canCourseBeDeleted(
            extractCorrelationId(headers).toString(),
            extractCustomerId(headers).toString(),
            courseId)
            .blockingSingle();
```

```
public CanCourseBeDeletedInfo canBeDeletedInfo() {
    return new CanCourseBeDeletedInfo(programs.isEmpty(), programs);
}
```

Adding user to group

Mina kurser och program

Grunderna i GDPR

Baloo Learning bjuder på en kurs i ...

STARTA KURSEN

Checklistan som ...

Här har vi samlat 10 kurser dä...

VISA PROGRAM

When a user is added to a group, courses targeted to that group should be targeted to the user.

When listing courses. We could fetch the groups for the user, and then all targeted courses for thoose groups.

Adding User to group

When a user is added to a group, courses targeted to that group should be targeted to the user.

```
List<Plan> plansInGroup = groupPlanRepositoryFactory.get(customerId).findPlans((event.groupId));
Coworker coworker = coworkerTargetedCoursesRepositoryFactory.get(customerId).find(event.userId).orElse(new Coworker(event.userId));

plansInGroup.forEach(plan -> {
    List<Event> events = coworker.addPlan(event.groupId, plan.planId, new Content(plan.content.contentId, plan.content.contentType));
    events.forEach(planAddedEvent -> publisher.publish(new PayloadMetaData(customerId, planAddedEvent.eventType), planAddedEvent));

});
```

Adding User to group

When listing courses. We could fetch the groups for the user, and then all targeted courses for those groups.

```
@GET
@Path("/coworkers/{coworkerId}/planned-content")
public Response getPlanedContent(@PathParam("customerId") CustomerId customerId, @PathParam("coworkerId") String coworkerId) {
    logger.info("Getting planned-content for coworker [{}]", coworkerId);
    Optional<Coworker> coworkerTargetedCoursesOptional = coworkerRepository.get(customerId).find(coworkerId);
    ContentsDto contentsDto = ContentsDto.toDto(coworkerTargetedCoursesOptional.orElse(new Coworker(coworkerId)));
    logger.info("Returning published courses for coworker [{}]", coworkerId);
    return Response.ok(contentsDto).build();

}
```

Conclusion

- Makes us think.
- Encourages involvement of non developers
- Improves separation between services
- Sometimes frustrating to find correct names

Thank you