AUTHENTICATION, AUTHORIZATION, AND FINE-GRAINED ACCESS CONTROL

Andy Miller | amiller@objectpartners.com | @opiamiller |

OBJECT PARTNERS

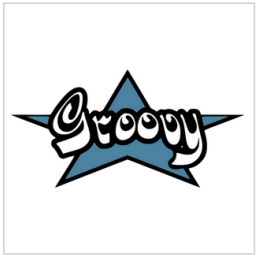
www.objectpartners.com

Front-End Applications

JavaScript framework expertise in technologies such as AngularJS and Bootstrap. AngularJS, Bootstrap, Jasmine, Grunt, Bower, npm

Web Applications

Rock solid expertise in Java, Groovy, Scala, Spring and Grails.



HAVE YOU EVER SEEN AN OLD PHOTO OF YOURSELF...

...AND BEEN EMBARRASSED AT THE WAY YOU LOOKED?

HAVE YOU LOOKED AT OLD CODE YOU WROTE...

```
PROGRAM MAIN                                |
INTEGER N, X                                |    SUBROUTINE SUB1(X,DUMSUB)
EXTERNAL SUB1                               |    INTEGER N, X
COMMON /GLOBALS/ N                          |    EXTERNAL DUMSUB
X = 0                                       |    COMMON /GLOBALS/ N
PRINT *, 'Enter number of repeats'          |    IF(X .LT. N)THEN
READ (*,*) N                                |      X = X + 1
CALL SUB1(X,SUB1)                           |      PRINT *, 'x = ', X
END                                         |      CALL DUMSUB(X,DUMSUB)
                                            |    END IF
                                            |    END
                                            |
```

... what the ****?

HAVE YOU LOOKED AT OLD PROJECT PLANS AND BUDGETS...

...AND BEEN EMBARRASSED AT HOW MUCH MONEY YOU SPENT ON THE THINGS THAT TODAY ARE SO EASY?

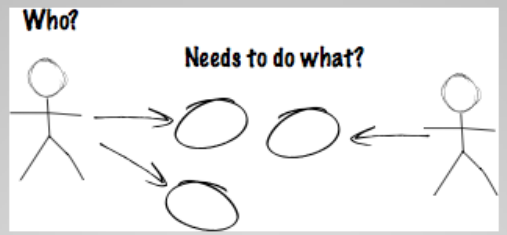
THE TRUTH IS WE SPENT ENTIRE PROJECT BUDGETS ON WHAT, THEN, WAS HARD STUFF...

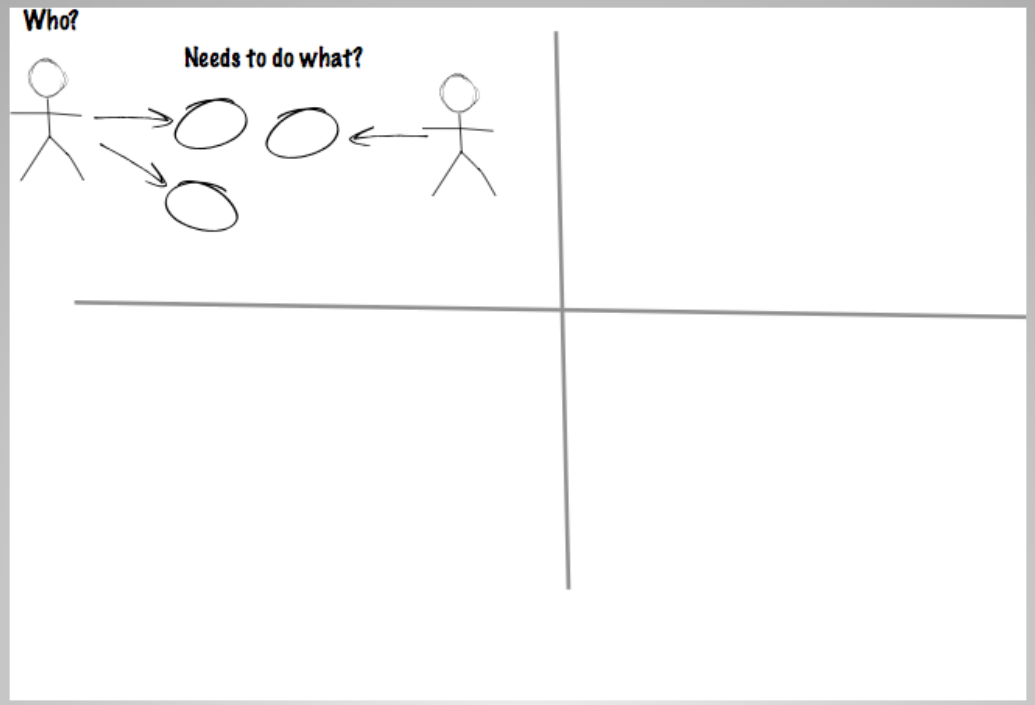
- figuring out how to organize source code
- figuring out dependency management
- figuring out build automation
- figuring out how to save data in a database
- figuring out test automation
- ...

Looking back, it's amazing we had time (or money) left over to work on the important things...

LIKE WHAT THE SOFTWARE WAS ACTUALLY SUPPOSED TO DO!

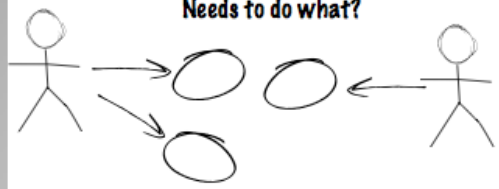
WHEN BUILDING A SYSTEM...




Who?
Needs to do what?

Who?

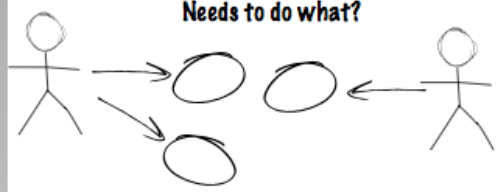
Needs to do what?



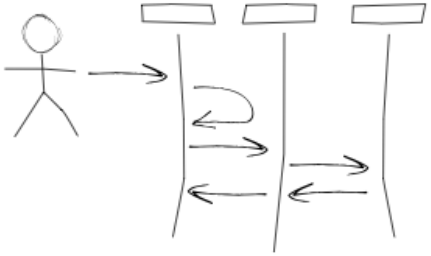
How's the system gonna do that?

Who?

Needs to do what?

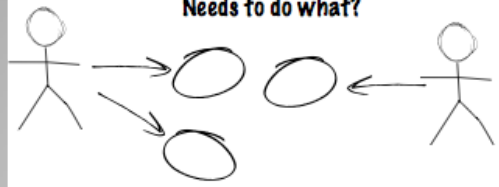


How's the system gonna do that?



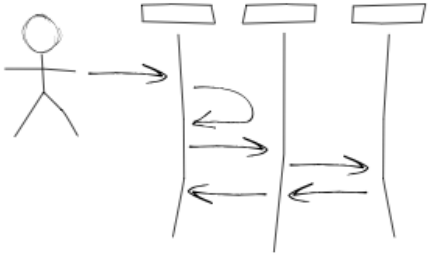
Who?

Needs to do what?



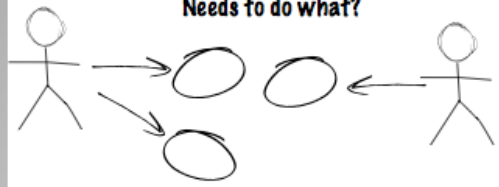
What are the elements of the system?

How's the system gonna do that?

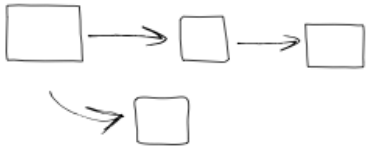


Who?

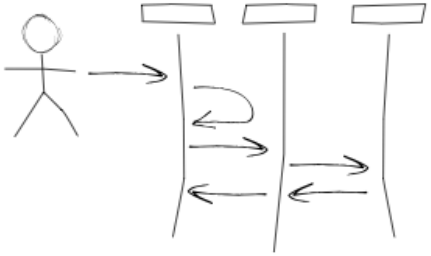
Needs to do what?



What are the elements of the system?

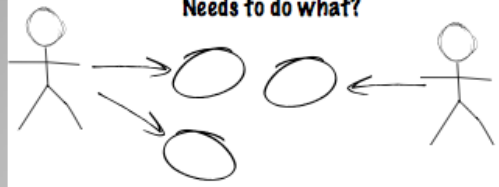


How's the system gonna do that?

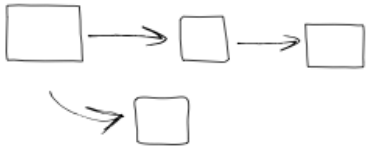


Who?

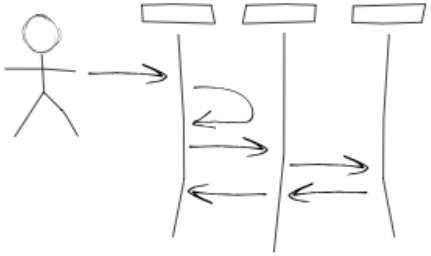
Needs to do what?



What are the elements of the system?



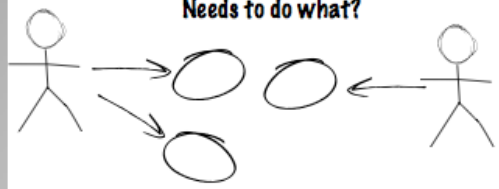
How's the system gonna do that?



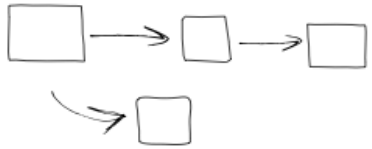
What's the priority and schedule?

Who?

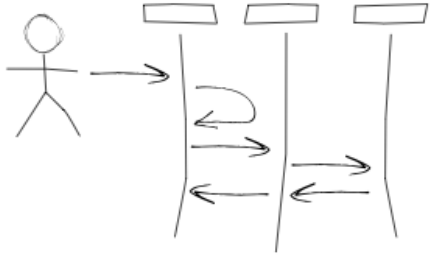
Needs to do what?



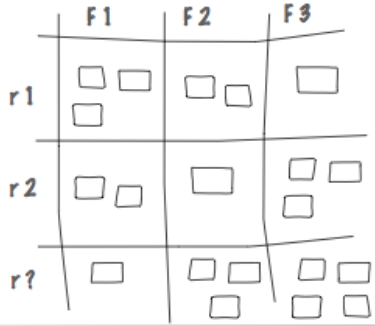
What are the elements of the system?



How's the system gonna do that?



What's the priority and schedule?



PATTERN LANGUAGES

WE'VE "SOLVED" SOME STUFF...

- structuring the vertical slice
- organizing source code
- dependency management
- build automation
- persistence
- schema evolution
- test automation

BUT SOME THINGS ARE STILL HARD...

- Requirements management
- Integrating with other systems
- Asynchronous processing
- Application security
- Batch processing
- Functional testing
- Working with money
- ...

SOME THINGS ARE JUST "HARD"...

- Requirements management
- Integrating with other systems
- Asynchronous processing
- **APPLICATION SECURITY**
- Batch processing
- Functional testing
- Working with money
- ...

APPLICATION SECURITY

0. anonymous (no security)
1. authentication of multiple users
2. authentication of multiple tenants
3. user authentication AND user authorization
4. fine grained user authorization varying across resources
5. data security (at the query level)
6. api security

These are complex, multi-dimensional problems. Cool! I like complex problems, but they're only fun to solve once (or maybe two or three times.)

So let's look a little deeper...

PATTERN LANGUAGE

security

---

anonymous | authorized | restricted | unrestricted
secured | unsecured

---

authentication | username | password

---

roles/permissions/authorities

---

fine grained security

API | API consumer | bearer token

USERS & ROLES

- Anonymous : No security necessary.
- Single User : one step beyond "anonymous". Address this situation by using the @Secured annotation (provided by the spring-security-core plugin) to secure controller actions.

```
@Secured(["isFullyAuthenticated()"])
```

- Multipe Users : As soon as you've secured an application you'll discover you can do things with the knowledge of who's logged in. Things like audit logging are possible (who did what/when.)
- Multipe Users / Multiple Roles

USERS & ROLES

- Multipe Users / Multiple Roles: With multiple users you'll find that you want to allow some users to do somethings and other users to do other things. This too can be handled with the @Secured annotation

```
@Secured(["ROLE_SUPERUSER"])
def rmMinusRStar(String pathname) {
    ...
}

@Secured(["ROLE_MOM"])
def makeGoodCookies(Integer count) {
    (count * 2).times {
        ...
    }
}
```

MULTIPLE TENANTS

- multiple users
- multiple resources
- each resource is owned by a single user
- "OWNER" can grant access to other users

Eventually we're going to need to store data for multiple customers/tenants in the same database.

HOW DO WE HANDLE THIS?

Roles are great for implementing details like "Only the superuser can run with scissors" and "Never let Mom touch this feature!"

But roles can't capture variation across resources.

Fine-grained Access Control

==

Variation of access across users and resources

FINE-GRAINED ACCESS CONTROL

I want a simple way to discuss this with product ownsers

...so let's extend the pattern language.

and

I want the implementation to be just as simple...
no room for stupid coding mistakes!

...so let's build another annotation.

LEVELS OF AUTHORIZATION

Class level authorization: grant users ability to create new instances

---

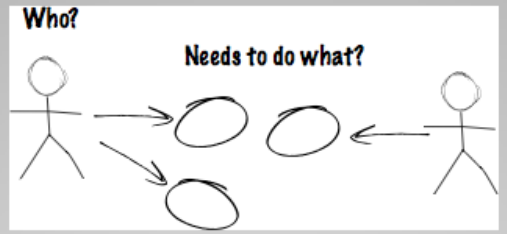
Instance level authorization: allow users to work with individual domain objects

```
@Secured(["isFullyAuthenticated()"])
def show(Project org) {
    ...
}
```

```
@Secured(["isFullyAuthenticated()"])
@Authorized( clazz = Project, idParam = "id", permission="VIEWER" )
def show(Project org) {
    ...
}
```

WHEN "ROLES" VARY ACROSS MULTIPLE TENANTS

0. anonymous (no security)
1. authentication of multiple clients (bearer tokens to secure a rest api)
2. authentication of multiple users (spring security to restrict access to ctrl/action)
3. user authentication AND user authorization (spring security "authorities" to restrict acces to ctrl/action)
4. user authorization that varies across different data elements == fine grained security
5. data security (at the query level)


Who?
Needs to do what?

FINE GRAINED SECURITY EXAMPLE

1. I (a user) can create new resources
2. I "own" resources that I've created
3. I am a "VIEWER" for some resources
4. I can grant "OWNER", "VIEWER", and so on to other users

usage example: class level authorization

```
@Authorized( clazz = Project, permission="OWNER" )
def create() {
    ...
}

@Authorized( clazz = Project, permission="OWNER" )
def save() {
    ...
}
```

usage example: instance level authorization

```
@Authorized( clazz = Project, idParam = "id", permission="OWNER" )
def edit(Project project) {
    render template: "edit", model: [project: project]
}

@Authorized( clazz = Project, idParam = "id", permission="OWNER" )
def update(Project project) {
    ...
}
```

usage example: list of authorized instances

```
def list() {
    def user = springSecurityService.currentUser
    def projectList = authorizationService.authorizedInstances(
                          user?.id,
                          Project,
                          params
                      )
    def projectTotal = authorizationService.authorizedInstanceCount(
                           user?.id,
                           Project
                       )
    [ projectList: projectList, projectTotal: projectTotal, ]
}
```

IMPLEMENTATION...

github.com/onetribeyoyo/grails-domain-authorization

- `@Authorized` annotation
- `DomainAuthorizationFilters`
- `AuthorizationService`
- `Authorization` (domain object)
- `AuthorizationTagLib`

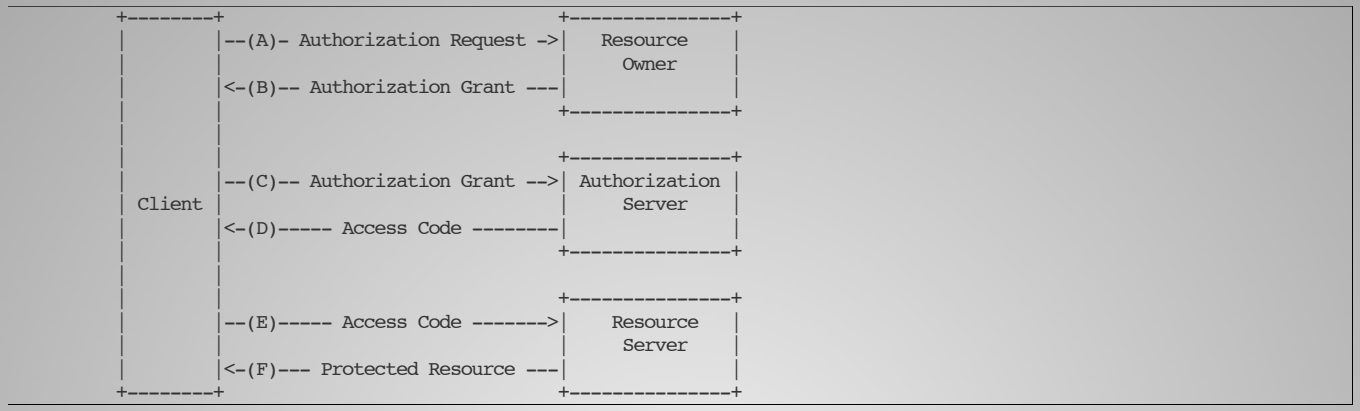
API SECURITY

A REST API can be secured with a two part process...

1. authentication is handled with an OATH 2 style process
2. and authorization is handled with @Authorized annotation

API SECURITY

Figure 1 from **self-issued.info/docs/draft-ietf-oauth-v2-bearer.html** sums it up nicely...



Note: Web app handles all three server-side roles.

```
@Secured(["IS_AUTHENTICATED_ANONYMOUSLY"])
@SecuredApi
def apiGet(String projectCode) {
    ....
}
```

```
ApiConsumer {

    boolean validateResource(String rsrc) {
        if (!rsrc?.trim()) {
            return false
        } else if (!isActive()) {
            return false
        } else if (!scope) {
            return false
        } else {
            return scope.find { pattern -> matchUriPattern(rsrc, pattern) }
        }
    }
```

BEARER TOKEN?

"authorization" http request header field formatted like...

```
"bearer b64token"
```

where

```
b64token = 1*( ALPHA / DIGIT / "-" / "." / "_" / "~" / "+" / "/" ) *"="
```

```
ApiConsumerSpec {



CODE EXAMPLES
```

PATTERN LANGUAGE

The words we use to discuss difficult topics should be the same when we talk about requirements, talk about technical concepts, and when we talk about code.

QUESTIONS?

Slides + Src: **github.com/onetribeyoyo/dev-objecttives-2015**

grails-domain-authorization plugin
github.com/onetribeyoyo/grails-domain-authorization

magic-task-machine (example app)
github.com/onetribeyoyo/mtm

Andy Miller | **amiller@objectpartners.com** | **@opiamiller** |