IQSS
The Institute for Quantitative Social Science

The Dataverse Project

BOF4161:

REST in Peace with Java EE

Gustavo Durand
Institute for Quantitative
Social Science
Harvard University

Michael Bar-Sinai
Ben-Gurion University, Israel
Institute for Quantitative
Social Science
Harvard University

@dataverseorg

Session Description

This sessions describes an efficient and secure implementation of a REST API for a large Java EE 7 web application. Dataverse, Harvard’s popular institutional scientific data repository, sports a comprehensive REST API which shares much functionality with the application's UI. Implementing such an API raises many engineering challenges, from code reuse and security to JSON roundtrips and proper HTTP response codes. To meet these challenges, the Dataverse development team defined new annotations, gave a modern twist to the classic Command pattern, used exceptions in a novel way, and other such niceties. After more than a year in production, the speakers feel confident enough to share this design with the “REST” of the Java community.

The Dataverse Project

- Began in 2006
- Software framework for publishing, citing and preserving research data
- Open source on GitHub for others to install
- Provides incentives for researchers to share:
  - Recognition & credit via data citations
  - Control over data & branding
  - Fulfill Data Management Plan requirements

Dataverse Team

Team:

Number of developers has varied over years. Currently:

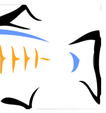
- 1 project manager
- 1 technical lead
- 4 dedicated developers
- Shared QA
- Shared UI designer

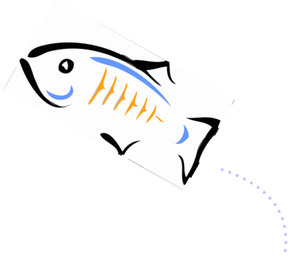
Community:

- Working with several partners to add features important to their installations
- Part-time developers

Dataverse Technology

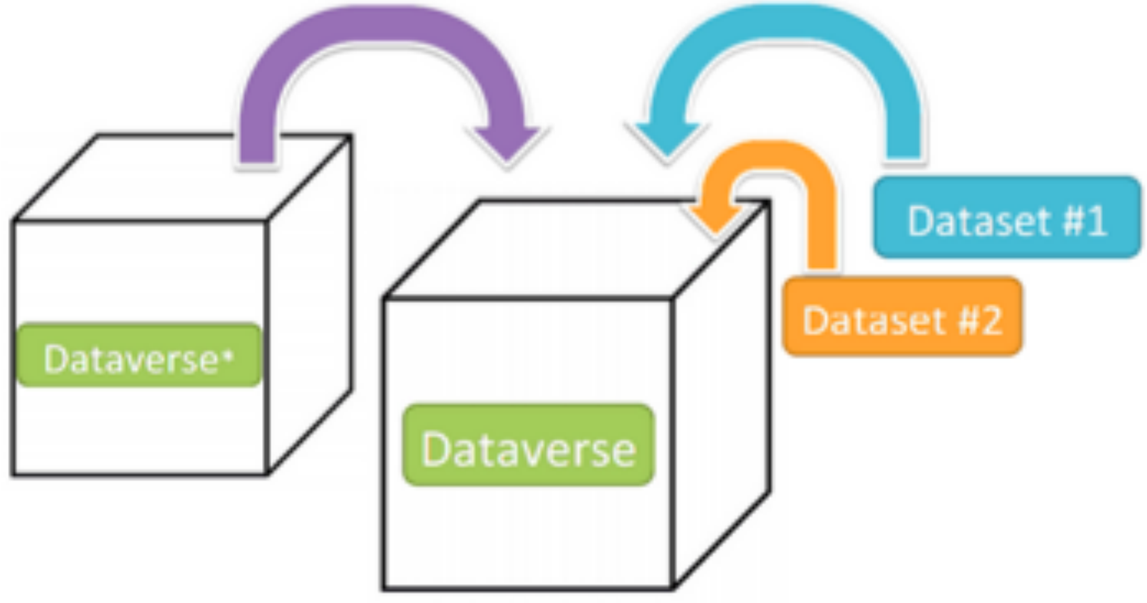
- Glassfish Server 4.1
- Java SE8
- Java EE7
- Presentation: JSF (PrimeFaces), RESTful API
- Business: EJB, Transactions, Asynchronous, Timers
- Storage: JPA (Entities), Bean Validation
- Storage: Postgres, Solr, File System





What is a **Dataverse** or **Dataset**?

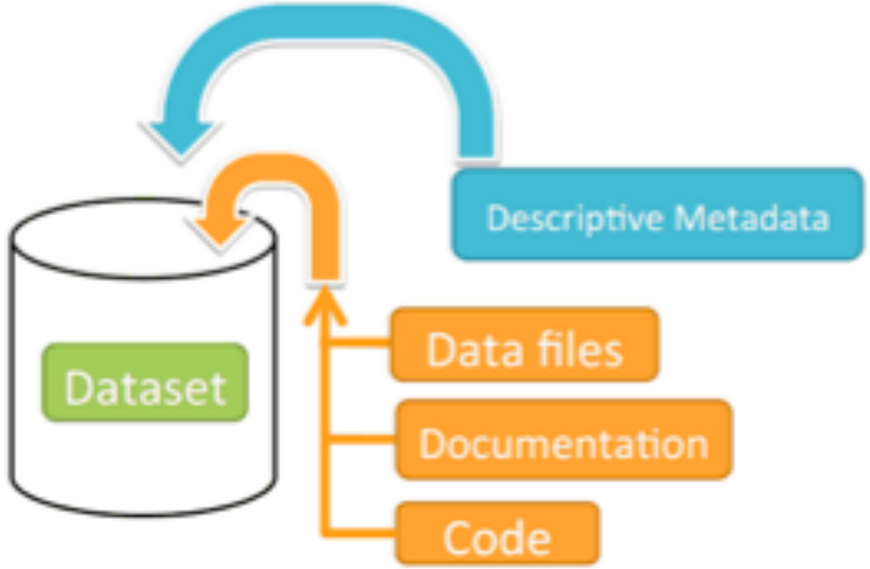
Schematic Diagram of a **Dataverse** in Dataverse 4.0



Container for your **Datasets** and/or **Dataverses***

* Dataverses can now contain other Dataverses (this replaces Collections & Subnetworks)

Schematic Diagram of a **Dataset** in Dataverse 4.0



Container for your data, documentation, and code.

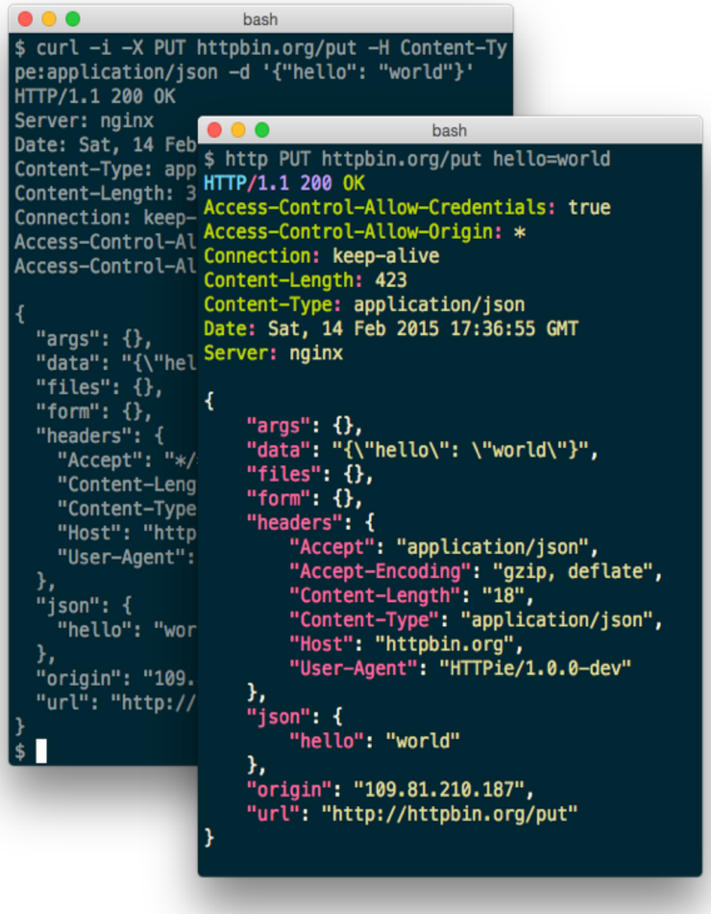
Image created by: Eleni Castro

Permissions, Roles, RoleAssignees

- **Permissions**
  - Based on DVObject
- **Roles**
  - Grouping of Permissions
- **Role Assignees**
  - Users
  - Groups: Explicit groups, Dynamically defined groups
- **Role Assignments**
  - Subject: `RoleAssignnee`
  - Verb: Role
  - Object: DefinitionPoint (`DVObject`)

Tools

- **curl**
  - A command line tool for transferring data with URLs
- **HTTPie**
  - A command line HTTP client that allows for sending arbitrary HTTP requests using a simple and natural syntax, and displays colorized output
- **jq**
  - A lightweight and flexible command-line JSON processor



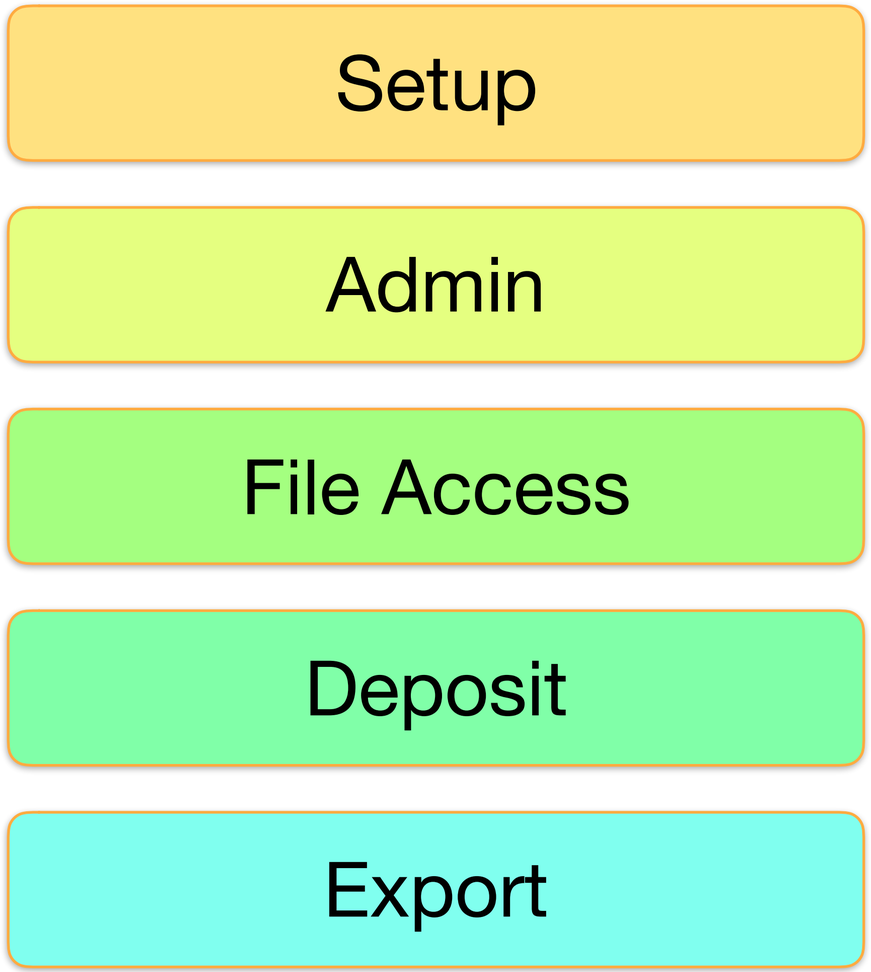
Why APIs?

- Design App as a *platform*
- Setup application state
  - Pre-populating a fresh dev install

    e.g. after dropping the db
  - Pre-populating test cases
- Inspect application state without the UI

  which might not be there
- Foster a developer community
- Allow for integration tests

Dataverse uses of APIs



Design Considerations

- Request / Response formats
  - XML vs JSON
- URL Design
  - Consistency
- API Versioning
  - Allows backward compatibility
  - Calls to no version default to latest version

Design Considerations

It is important to make these decisions before you *publish* the API!

- Request / Response formats
  - XML vs JSON
- URL Design
  - Consistency
- API Versioning
  - Allows backward compatibility
  - Calls to no version default to latest version

What Java EE 7 provides (JAX-RS)

ANNOTATIONS!

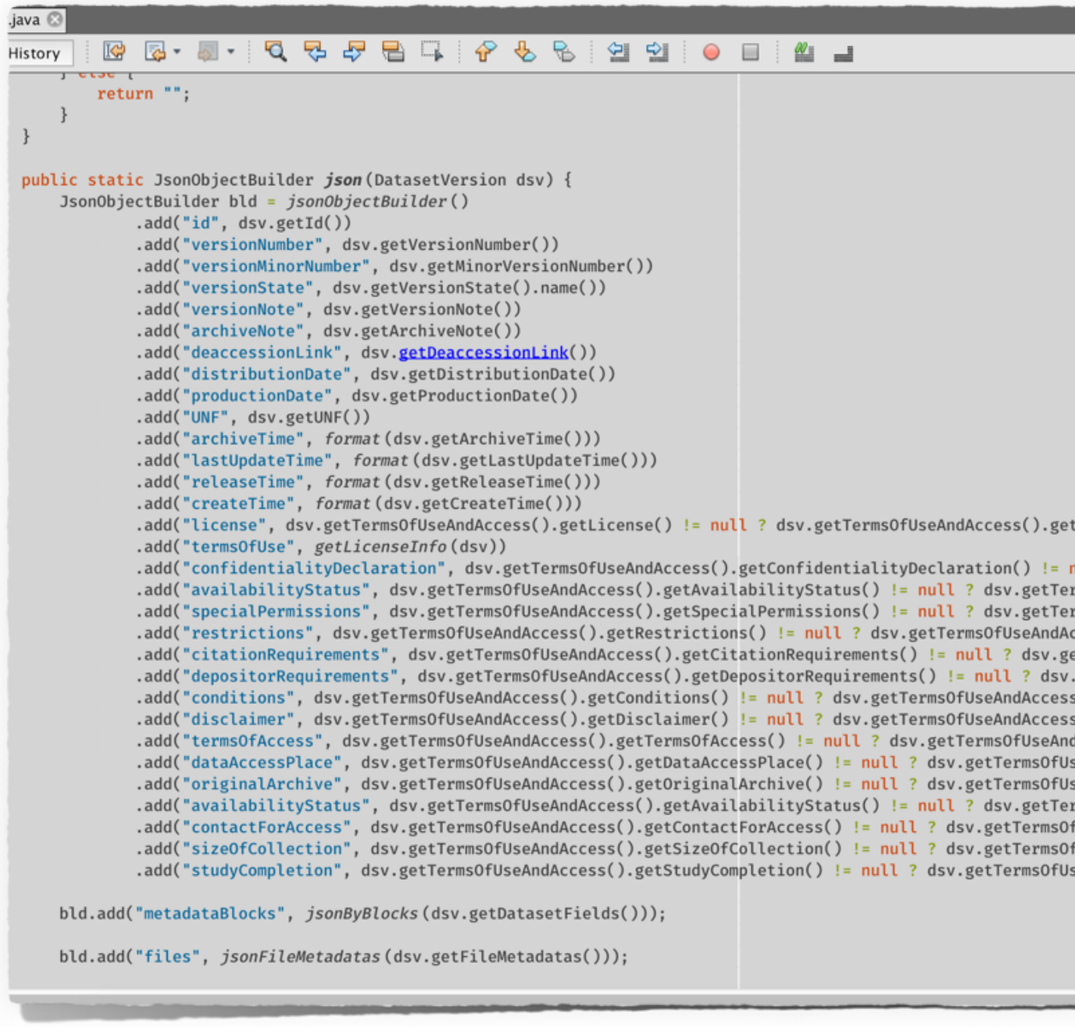
- `@Path` specifies the relative path for a resource class or method.
- `@GET`, `@PUT`, `@POST`, `@DELETE` and `@HEAD` specify the HTTP request type of a resource.
- `@Produces` specifies the response Internet media types (used for content negotiation).
- `@Consumes` specifies the accepted request Internet media types.
- `@PathParam` binds the method parameter to a path segment.
- `@QueryParam` binds the method parameter to the value of an HTTP query parameter.

A Simple Example

```
@Path("datasets")
public class Datasets extends AbstractApiBean {
    ...
    @GET
    @Path("/export")
    @Produces({"application/xml", "application/json"})
    public Response exportDataset(@QueryParam("persistentId") String persistentId,
                                  @QueryParam("exporter") String exporter) {
        ...
    }
}
```

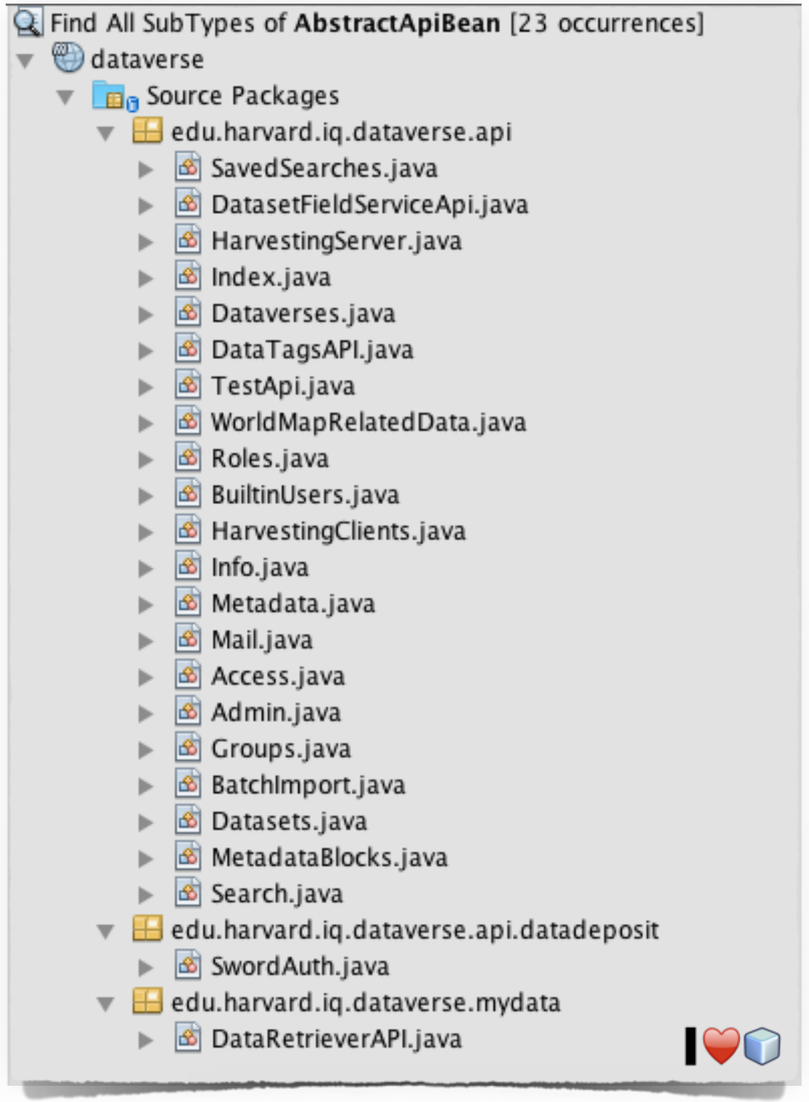
JSON: There and Back Again

- Automatic vs. Manual: We went with **manual**
  - Representation has to be stable, or you break your user's code
  - Objects can have different representation in different contexts
  - Recursive structures are tricky
- Implementation *uses static methods* to better integrate with client code
- Two "printer" classes: full and brief
- One parser class
- Tedious to write, OK to maintain.
- Alleviated by null-safe object builders and JSON stream collectors (yay Java 8 streams!)



AbstractApiBean

- Base class for all API resource classes.
- Code reuse
  - Useful injections (e.g. `HttpServletRequest`)
  - Common Beans
  - Finding user by API token
- **Support of a DSL-ish environment for writing API endpoints**.
  - HTTP response codes methods
    - `okResponse(…)`, `notFound(…)`
    - Inspired by *Play!2.0*
  - `findUser()`, `findDataverse()`, `findDvo()`



RoleApiBean extends AbstractApiBean

```
@GET
@Path("{id}")
public Response viewRole( @PathParam("id") Long id) {
    try {
        DataverseRole role = rolesSvc.find(id);
        if ( role == null ) {
            return notFound("role with id " + id + " not found");
        } else  {
            return ( permissionSvc.userOn(findUserOrDie(),role.getOwner())
                                  .has(Permission.ManageDataversePermissions) ) ?
                        okResponse( json(role) )
                        : permissionError("Permission required to view roles.");
        }
    } catch (WrappedResponse ex) {
        return ex.getResponse();
    }
}
```

RoleApiBean extends AbstractApiBean

```
@GET
@Path("{id}")
public Response viewRole( @PathParam("id") Long id) {
    try {
        DataverseRole role = rolesSvc.find(id);
        if ( role == null ) {
            return notFound("role with id " + id + " not found");
        } else  {
            return ( permissionSvc.userOn(findUserOrDie(),role.getOwner())
                                  .has(Permission.ManageDataversePermissions) ) ?
                        okResponse( json(role) )
                        : permissionError("Permission required to view roles.");
        }
    } catch (WrappedResponse ex) {
        return ex.getResponse();
    }
}
```

Static Imports

RoleApiBean extends AbstractApiBean

```
@GET
@Path("{id}")
public Response viewRole( @PathParam("id") Long id) {
    try {
        DataverseRole role = rolesSvc.find(id);
        if ( role == null ) {
            return notFound("role with id " + id + " not found");
        } else  {
            return ( permissionSvc.userOn(findUserOrDie(),role.getOwner())
                                  .has(Permission.ManageDataversePermissions) ) ?
                        okResponse( json(role) )
                        : permissionError("Permission required to view roles.");
        }
    } catch (WrappedResponse ex) {
        return ex.getResponse();
    }
}
```

HTTP Responses

Static Imports

RoleApiBean extends AbstractApiBean

```
@GET
@Path("{id}")
public Response viewRole( @PathParam("id") Long id) {
    try {
        DataverseRole role = rolesSvc.find(id);
        if ( role == null ) {
            return notFound("role with id " + id + " not found");
        } else  {
            return ( permissionSvc.userOn(findUserOrDie(),role.getOwner())
                                  .has(Permission.ManageDataversePermissions) ) ?
                        okResponse( json(role) )
                        : permissionError("Permission required to view roles.");
        }
    } catch (WrappedResponse ex) {
        return ex.getResponse();
    }
}
```

HTTP Responses

Static Imports

Wrapped Responses

Problem:

Unlike UI, API has no session; it has to build context for each request.

UI

1. Login (gets user)
2. Navigate to object (gets object)
3. Check permissions and select operation (action)
4. Actually have the user operate on the object

API

1. Get user, get object, check permission and operate on object.
   Repeat. Each. call.

The Code is Loooong

```
@GET
@Path("{identifier}/groups/{aliasInOwner}")
public Response getGroupByOwnerAndAliasInOwner(
                            @PathParam("identifier") String dvIdtf,
                             @PathParam("aliasInOwner") String grpAliasInOwner){
    Dataverse dv = findDataverse(dvIdtf);
    if ( dv == null ) {
      return notFound( ... );
    }
    User user = findUser();
    if ( user == null ) {
      return notAuthorized( ... );
    }
    if ( ! Permissions.check(user, dv, SEE_GROUPS) ) {
      return notAuthorized( ... );
    }
    Group group = findExplicitGroup(dv, grpAliasInOwner );
    if ( group == null ) {
      return notFound( ... );
    } else {
      return okResponse( json(group) );
    }
}
```

/!\ Not a production code

The Code is Loooong

```
@GET
@Path("{identifier}/groups/{aliasInOwner}")
public Response getGroupByOwnerAndAliasInOwner(
                            @PathParam("identifier") String dvIdtf,
                             @PathParam("aliasInOwner") String grpAliasInOwner){
    Dataverse dv = findDataverse(dvIdtf);
    if ( dv == null ) {
      return notFound( ... );
    }
    User user = findUser();
    if ( user == null ) {
      return notAuthorized( ... );
    }
    if ( ! Permissions.check(user, dv, SEE_GROUPS) ) {
      return notAuthorized( ... );
    }
    Group group = findExplicitGroup(dv, grpAliasInOwner );
    if ( group == null ) {
      return notFound( ... );
    } else {
      return okResponse( json(group) );
    }
}
```

Get, test,
return if fail

/!\ Not a production code

The Code is Loooong

```
@GET
@Path("{identifier}/groups/{aliasInOwner}")
public Response getGroupByOwnerAndAliasInOwner(
                            @PathParam("identifier") String dvIdtf,
                             @PathParam("aliasInOwner") String grpAliasInOwner){
    Dataverse dv = findDataverse(dvIdtf);
    if ( dv == null ) {
      return notFound( ... );
    }
    User user = findUser();
    if ( user == null ) {
      return notAuthorized( ... );
    }
    if ( ! Permissions.check(user, dv, SEE_GROUPS) ) {
      return notAuthorized( ... );
    }
    Group group = findExplicitGroup(dv, grpAliasInOwner );
    if ( group == null ) {
      return notFound( ... );
    } else {
      return okResponse( json(group) );
    }
}
```

Get, test, return if fail

Validate permissions

/!\ Not a production code

Exceptions: the untold story

Exceptions: the untold story

Well, less told.

Exceptions: the untold story

Well, less told.

Requirement:

Exceptions: the untold story

Well, less told.

Requirement:

Given a dataverse ID, return its data, or an HTTP error.

Exceptions: the untold story

Well, less told.

Requirement:

Given a dataverse ID, return its data, or an HTTP error.

```
public Dataverse || HttpError findDataverse(String id);
```

Exceptions: the untold story

Well, less told.

Requirement:

Given a dataverse ID, return its data, or an HTTP error.

```
public Dataverse || HttpError findDataverse(String id);

final ?? result = findDataverse( id );
```

Exceptions: the untold story

Well, less told.

Requirement:

Given a dataverse ID, return its data, or an HTTP error.

```
public Dataverse || HttpError findDataverse(String id);

final ?? result = findDataverse( id );

public Dataverse findDataverse(String id) throws HttpError;
```

Exceptions: the untold story

Well, less told.

Requirement:

Given a dataverse ID, return its data, or an HTTP error.

```
public Dataverse || HttpError findDataverse(String id);

final ?? result = findDataverse( id );

public Dataverse findDataverse(String id) throws HttpError;
```

```
try {
    Dataverse aDataverse = findDataverse(id);
    // operate on aDataverse ...
} catch (HttpError err) {
    // Send proper HTTP error code ...
}
```

Exceptions allow methods to return multiple types*

* Terms and conditions:
  - All types except for one have to extend `java.lang.Exception`
  - Implemented as control flow
  - This view is less useful for cases where the exception propagates through the call stack until caught

WrappedResponse: an Exceptional Return

- Methods which may fail return the appropriate HTTP response, wrapped in a specialized exception
- Top method catches that exception and returns the wrapped response

```
private AuthenticatedUser findAuthenticatedUserOrDie( String key ) throws
WrappedResponse {
    AuthenticatedUser u = authSvc.lookupUser(key);
    if ( u != null ) {
      return u;
    }
    throw new WrappedResponse( badApiKey(key) );
}
```

```
protected <T> T failIfNull( T t, String errorMessage ) throws WrappedResponse {
    if ( t != null ) return t;
    throw new WrappedResponse(
                    errorResponse(Response.Status.BAD_REQUEST,errorMessage));
}
```

No Exceptions

```
@GET
@Path("{identifier}/groups/{aliasInOwner}")
public Response getGroupByOwnerAndAliasInOwner(@PathParam("identifier") String dvIdtf,
                                               @PathParam("aliasInOwner") String grpAliasInOwner ){
    Dataverse dv = findDataverse(dvIdtf);
    if ( dv == null ) {
      return notFound( ... );
    }
    User user = findUser();
    if ( user == null ) {
      return notAuthorized( ... );
    }
    if ( ! Permissions.check(user, dv, SEE_GROUPS) ) {
      return notAuthorized( ... );
    }
    Group group = findExplicitGroup(dv, grpAliasInOwner );
    if ( group == null ) {
      return notFound( ... );
    } else {
      return okResponse( json(group) );
    }
}
```

Wrapped Responses FTW!

```
@GET
@Path("{identifier}/groups/{aliasInOwner}")
public Response getGroupByOwnerAndAliasInOwner(@PathParam("identifier") String dvIdtf,
                                               @PathParam("aliasInOwner") String grpAliasInOwner ){
    try {
        return okResponse(json(
              findExplicitGroupOrDie(findDataverseOrDie(dvIdtf),
                                     createDataverseRequest(findUserOrDie()), grpAliasInOwner)));
    } catch (WrappedResponse wr) {
        return wr.getResponse();
    }
}
```

Wrapped Responses FTW!

200 OK

404 NOT FOUND (dataverse)

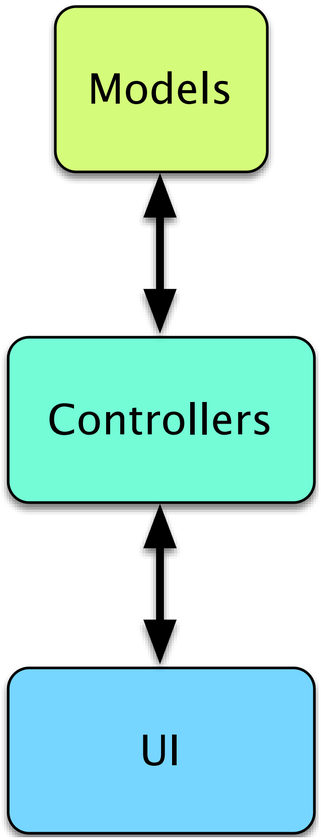
401 NOT AUTHORIZED (user not found)

```
@GET
@Path("{identifier}/groups/{aliasInOwner}")
public Response getGroupByOwnerAndAliasInOwner(@PathParam("identifier") String dvIdtf,
                                               @PathParam("aliasInOwner") String grpAliasInOwner ){

    try {
        return okResponse(json(
              findExplicitGroupOrDie(findDataverseOrDie(dvIdtf),
                                     createDataverseRequest(findUserOrDie()), grpAliasInOwner)));

    } catch (WrappedResponse wr) {
        return wr.getResponse();
    }
}
```

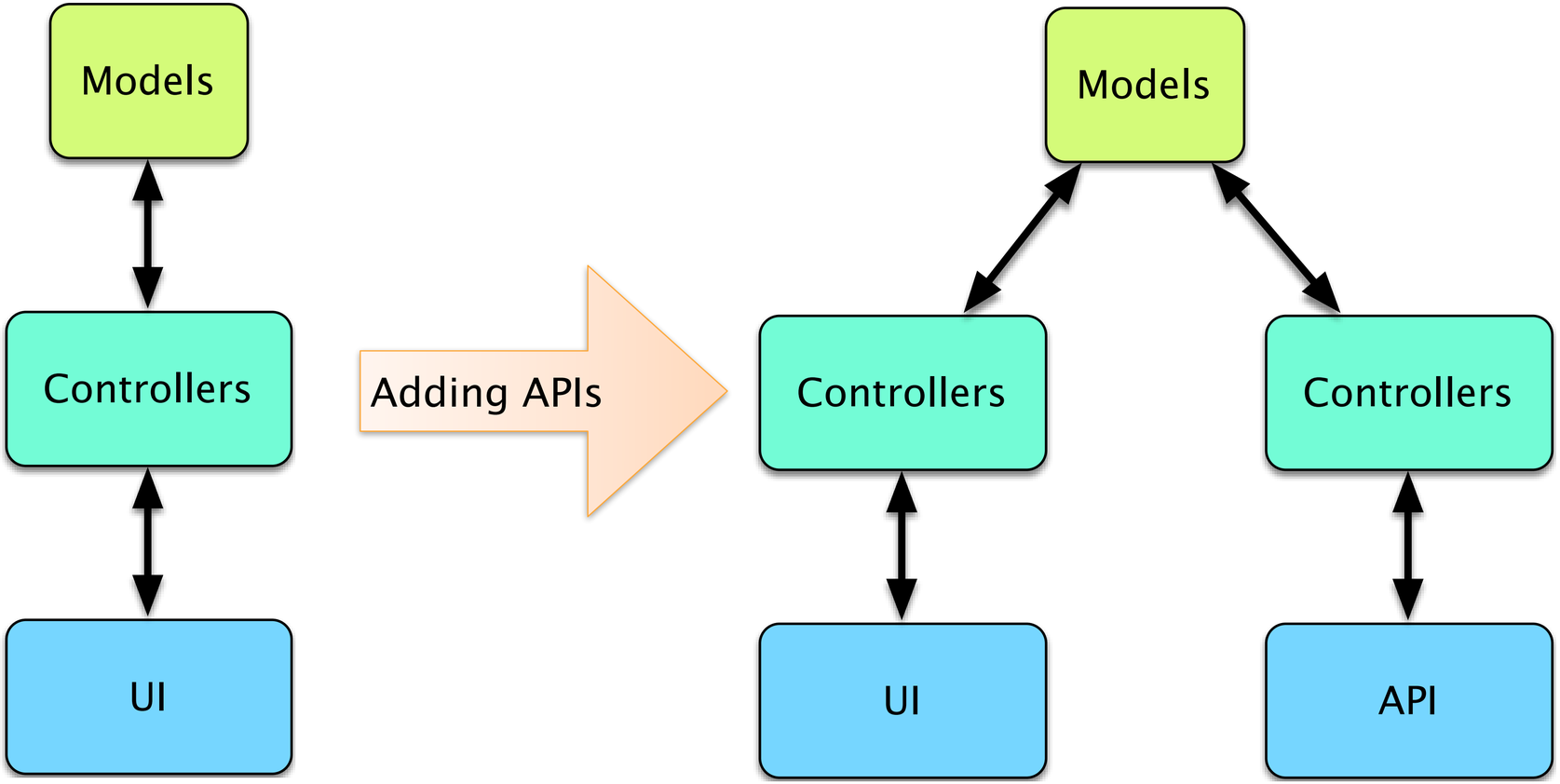
404 NOT FOUND (group)

401 NOT AUTHORIZED (user not authorized to new groups)

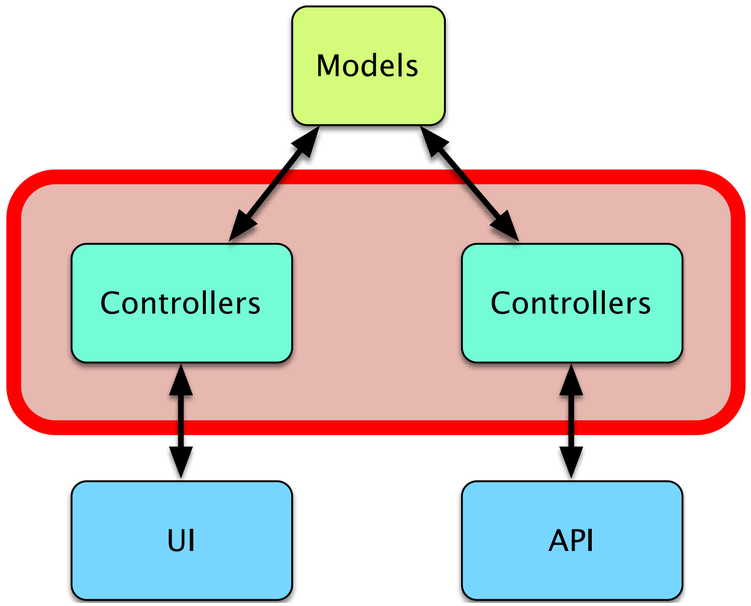
Problem:
Functionality Duplication



Problem:
Functionality Duplication



Problem: **Functionality Duplication**



- Functionality duplication
  - ~ Code duplication
- More permission validations
  - More chances of missing one

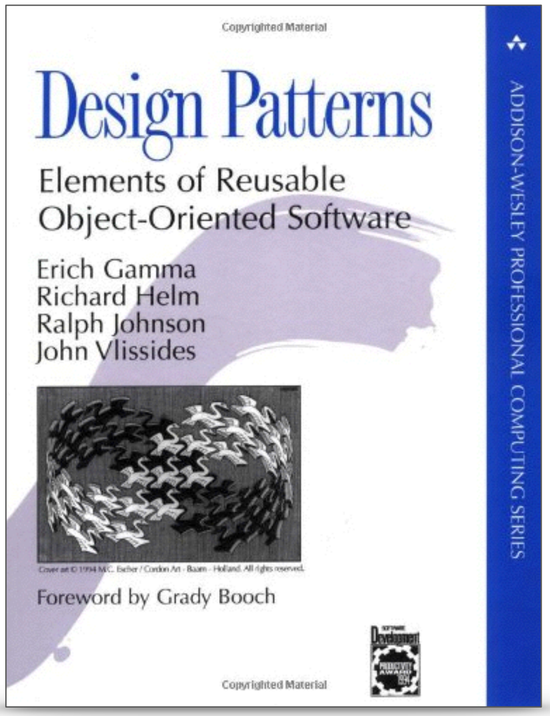
MVC alone is inefficient

for supporting a large API along a UI.

The Command Design Pattern

"Encapsulate a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations."

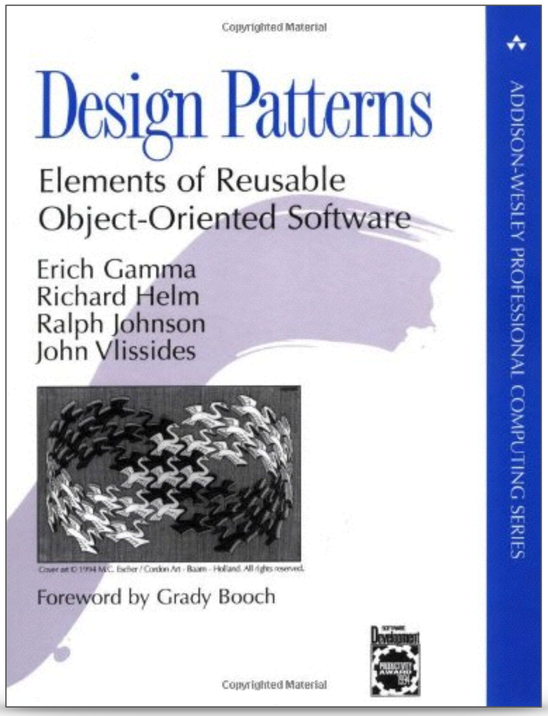
GoF, 1994



The Command Design Pattern

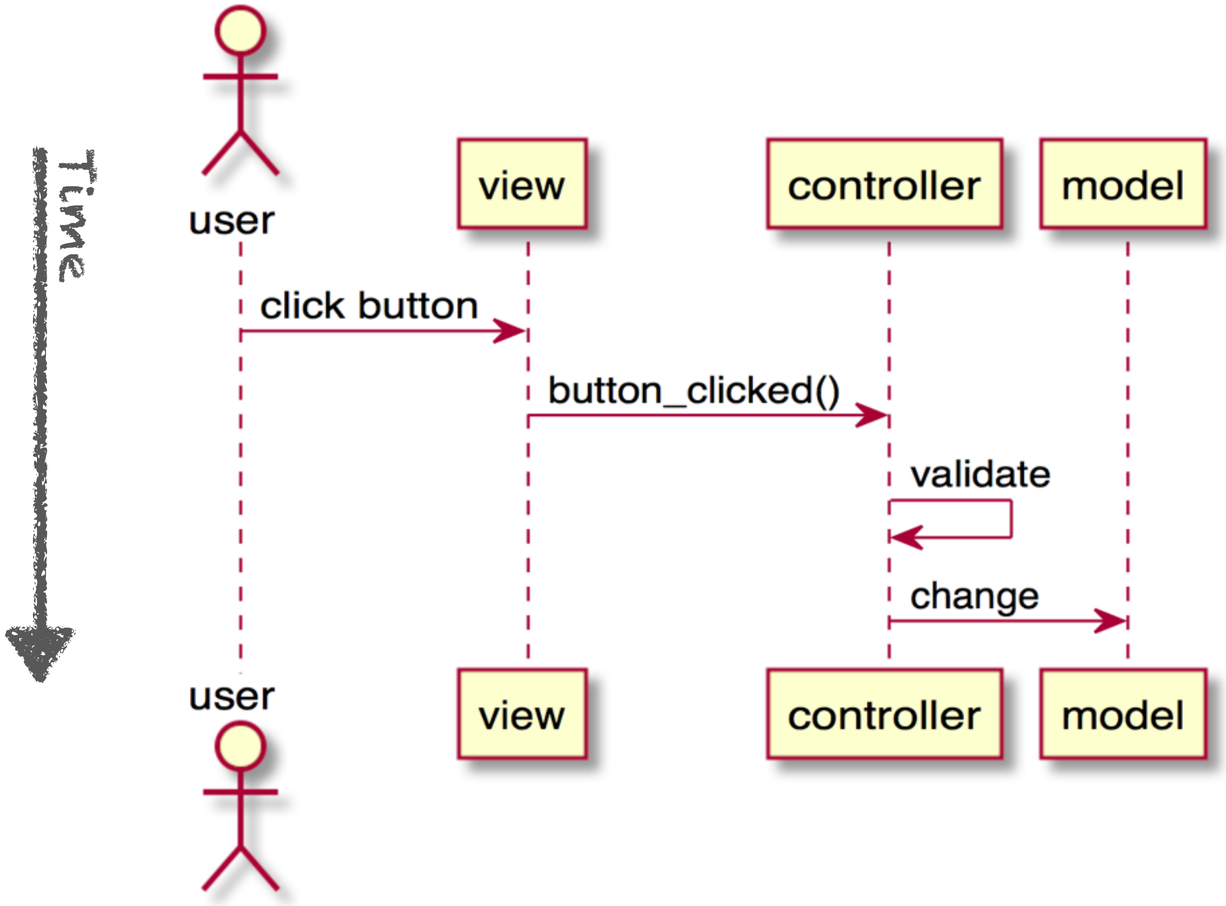
"Encapsulate a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations."

GoF, 1994

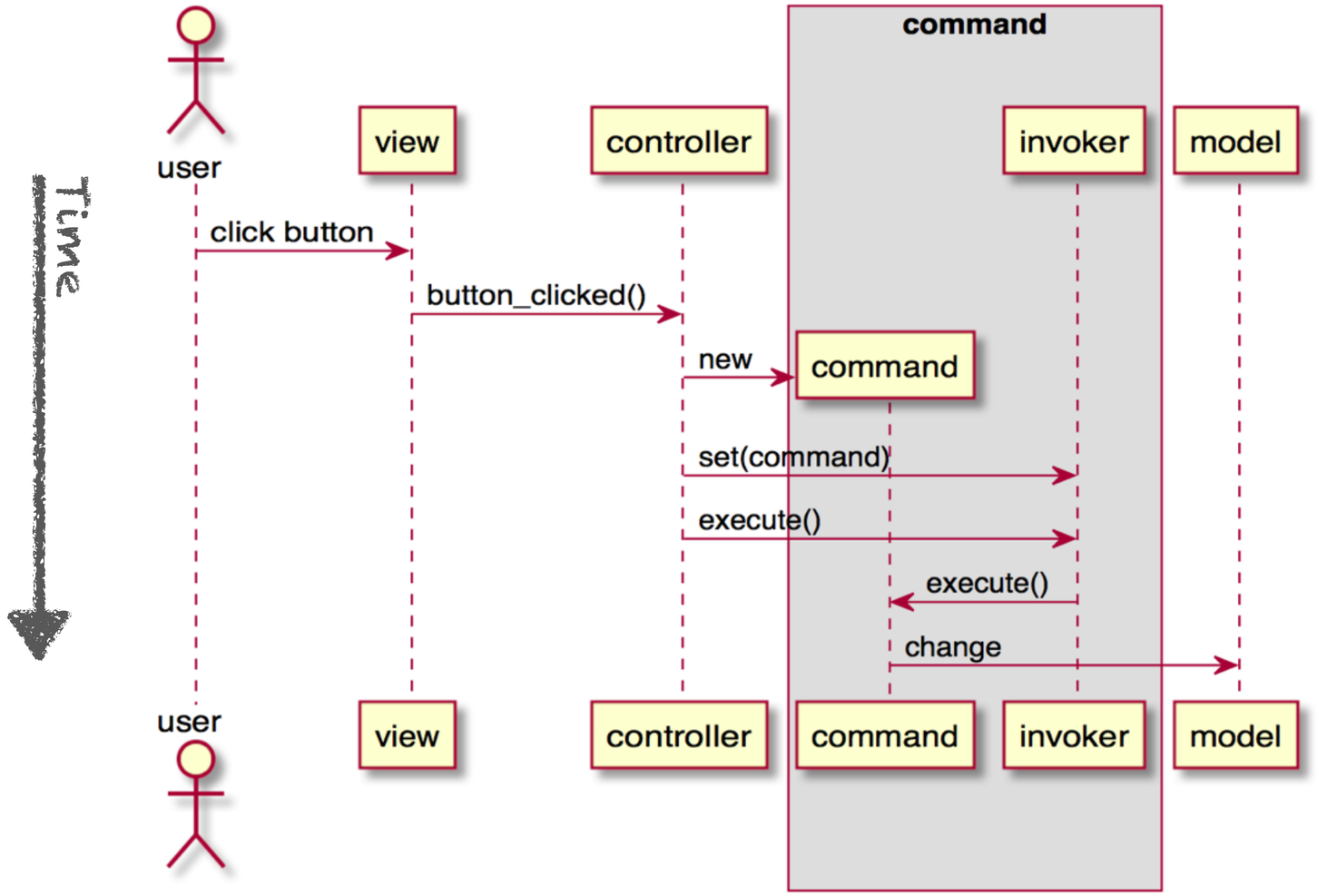


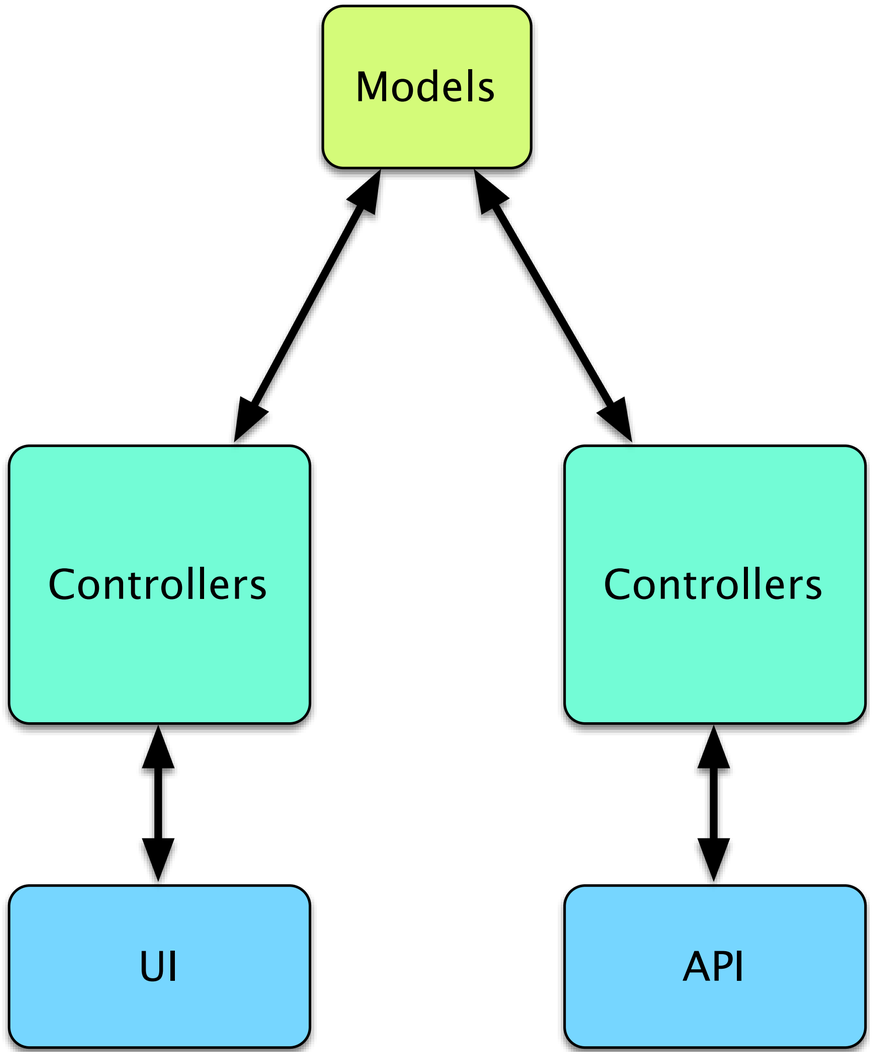
Using code as data

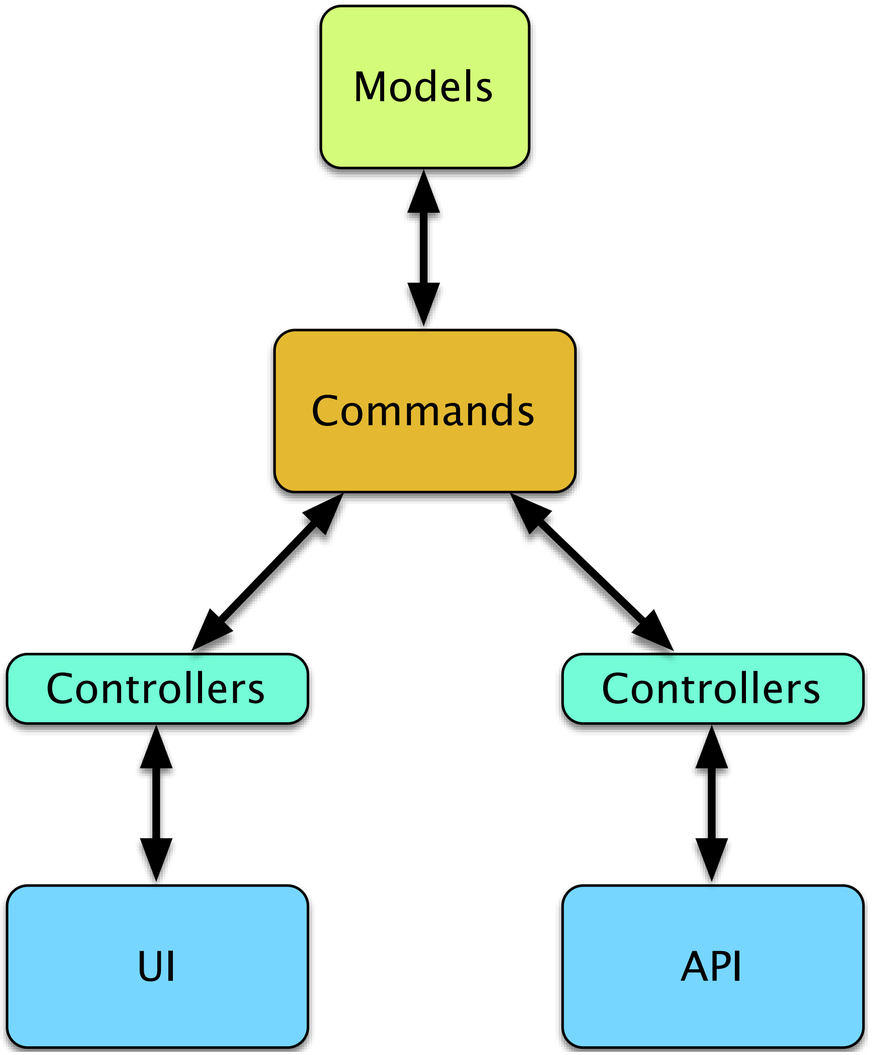
Classic Model-View-Controller



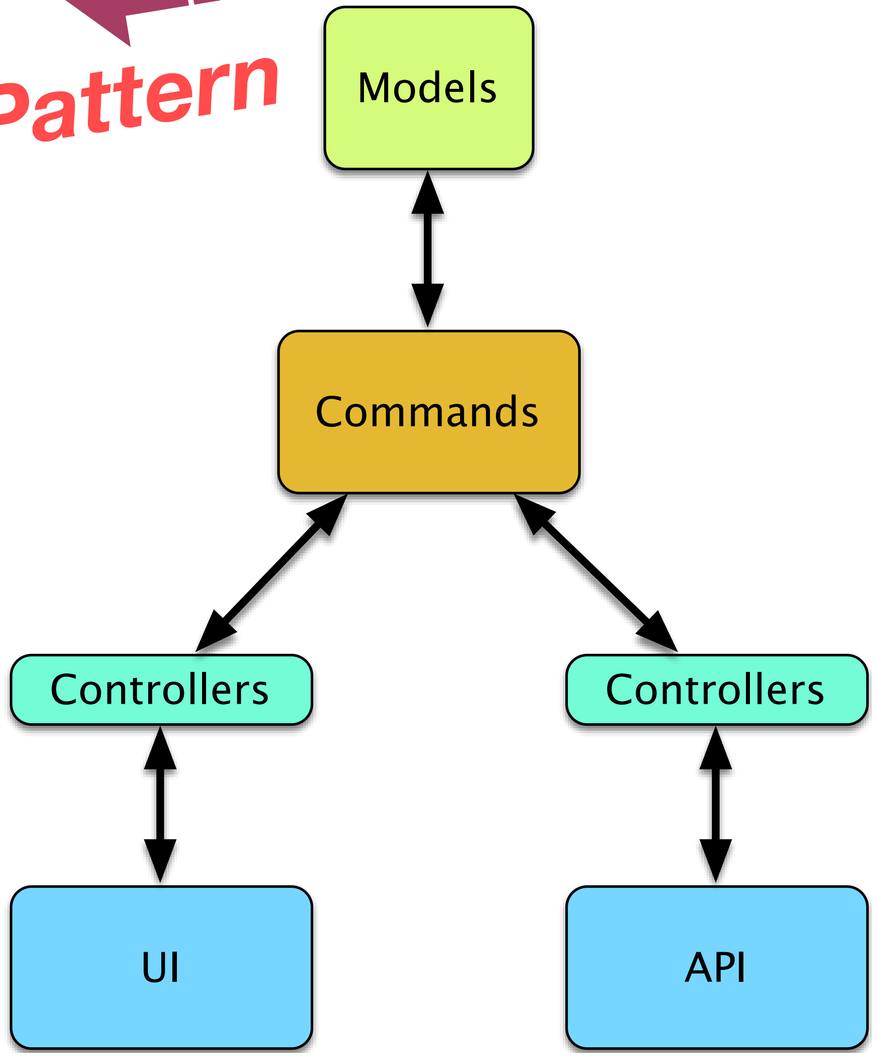
Classic Command Pattern

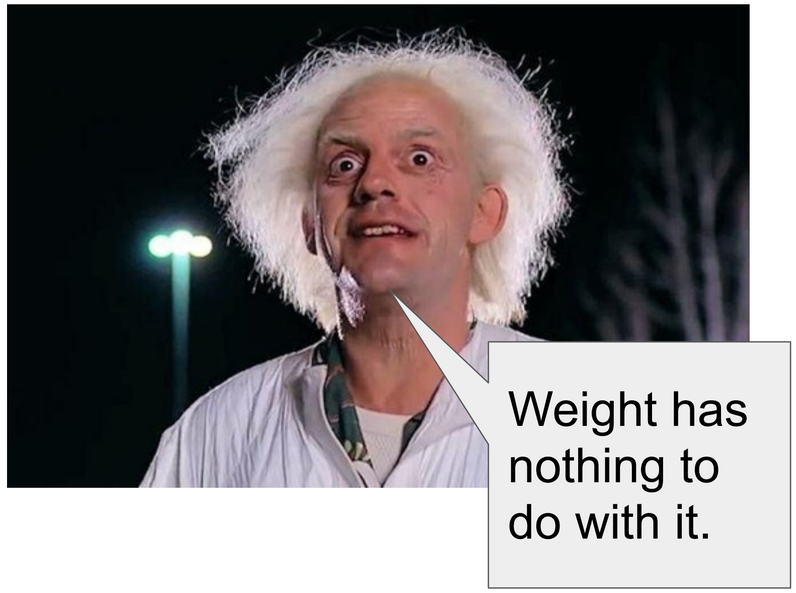



Models
Controllers
Controllers
UI
API


Models
Commands
Controllers
Controllers
UI
API

Back to the Command Pattern





Using the Command Design Pattern

Business logic implemented in Command Subclasses, Validation implemented in the “invoker” (engine)

- **Code Reuse**
  - Controllers become thinner, mostly preparing commands from view request and submitting them to the `CommandEngine` for execution
- **Security by Design**
  - Each command is annotated with required permissions
  - Permissions tested by the engine - single place, single implementation
  - Auditing: Engine logs all executed commands
- **Testable**
- **Standardized application operations**

Command, Adapted

```
public interface Command<R> {
  R execute( CommandContext ctxt ) throws CommandException;
  Map<String,DvObject> getAffectedDvObjects();
  Map<String,Set<Permission>> getRequiredPermissions();
  DataverseRequest getRequest();
}
```

- `execute()` is where the work is done.
  - Server resources and injected objects are made available via CommandContext
- `getAffectedDvObjects()` and `getRequiredPermissions()` detail which objects are affected and what permissions are needed to affect them
- `getRequest()` allows the permission system to detect which permissions the user has.

Sample Command: Delete a Role

```
@RequiredPermissions( Permission.ManageDataversePermissions )
public class DeleteRoleCommand extends AbstractVoidCommand {

    private final DataverseRole doomed;

    public DeleteRoleCommand(DataverseRequest aRequest, DataverseRole doomed ) {
        super(aRequest, doomed.getOwner());
        this.doomed = doomed;
    }

    @Override
    protected void executeImpl(CommandContext ctxt) throws CommandException {
        for (RoleAssignment ra:ctxt.roles().roleAssignments(doomed.getId())) {
            ctxt.roles().revoke(ra);
        }
        ctxt.roles().delete(doomed.getId());
    }
}
```

Sample Command Usage: Delete a Role

```
@DELETE
@Path("{id}")
public Response deleteRole( @PathParam("id") Long id ) {
  DataverseRole role = rolesSvc.find(id);
  if ( role == null ) {
    return notFound( "role with id " + id + " not found");
  } else  {
    try {
        execCommand( new DeleteRoleCommand(
                                createDataverseRequest(findUserOrDie()), role));
        return okResponse("role " + id + " deleted.");

    } catch (WrappedResponse ex) {
        return ex.refineResponse( "Cannot delete role " + id + "." );
    }
  }
}
```

Sample Command Usage: Delete a Role

```
@DELETE
@Path("{id}")
public Response deleteRole( @PathParam("id") Long id ) {
  DataverseRole role = rolesSvc.find(id);
  if ( role == null ) {
    return notFound( "role with id " + id + " not found");
  } else  {
    try {
        execCommand( new DeleteRoleCommand(
                                createDataverseRequest(findUserOrDie()), role));
        return okResponse("role " + id + " deleted.");

    } catch (WrappedResponse ex) {
        return ex.refineResponse( "Cannot delete role " + id + "." );
    }
  }
}
```

Results of `execCommad`

```
protected <T> T execCommand( Command<T> cmd ) throws WrappedResponse
```

Executing a command has 4 possible results:

- Issued command makes no sense (e.g. delete published data)
  - `IllegalCommandException`
  - Wrapped in WrappedResponse: 403 FORBIDDEN
- Issuing user is not permitted to perform the command
  - `PermissionException`
  - Wrapped in WrappedResponse: 401 UNAUTHORIZED
- General Server Error
  - `CommandException`
  - Wrapped in WrappedResponse: 500 INTERNAL_SERVER_ERROR
  - Plus, logging.
- Works
  - Return the result of the command (Java object)

Exec-ing a Command

```
@GET
@Path("{identifier}/facets/")
public Response listFacets( @PathParam("identifier") String dvIdtf ) {
    try {
       return okResponse(json(
                           execCommand(
                              new ListFacetsCommand(createDataverseRequest(findUserOrDie()),
                                                                           findDataverseOrDie(dvIdtf))
                )));
    } catch (WrappedResponse wr) {
        return wr.getResponse();
    }
}
```

Exec-ing a Command

200 OK
(JSON content)

500 INTERNAL SERVER ERROR
(oops, our bad)

401 UNAUTHORIZED
(user not found)

```
@GET
@Path("{identifier}/facets/")
public Response listFacets( @PathParam("identifier") String dvIdtf ) {
    try {
        return okResponse(json(
                            execCommand(
                                new ListFacetsCommand(createDataverseRequest(findUserOrDie()),
                                                      findDataverseOrDie(dvIdtf))

                )));
    } catch (WrappedResponse wr) {
        return wr.getResponse();
    }
}
```

403 FORBIDDEN
(for other cases)

401 UNAUTHORIZED
(user not permitted to list facets)

404 NOT FOUND
(dataverse)

Composable Commands

“If at first you don’t succeed, go get the public version”

```
@RequiredPermissions({})
public class GetLatestAccessibleDatasetVersionCommand extends AbstractCommand<DatasetVersion>{
  private final Dataset ds;
   ...
  @Override
  public DatasetVersion execute(CommandContext ctxt) throws CommandException {
      try {
          return ctxt.engine().submit(new GetDraftDatasetVersionCommand(getRequest(), ds));
      } catch(PermissionException ex) {
          return ctxt.engine().submit(new GetLatestPublishedDatasetVersionCommand(getRequest(),ds));
      }
  }
}
```

/!\ Code cleaned for clarity

Questions?

Thanks!

We thank the Frenkel Fund for taking part in funding this talk