Generical Controller

Design and Testing 2 – A++

Ángel Delgado Luna

Belén Garrido López

Ezequiel Portillo Jurado

Alejandro Rodríguez

Mª De Gracia Piñero Pastor

Index

1. Foundations

In last deliverables, we had to implement a lot of repetitive code in Controllers Section. From this deliverable, until the last one, we´ve decided to make a generical controller in order to set a common methodology when implementing this part. This will help us later to understand better what every team member is doing on his/her code.

2. Versions

As common in every technology or implementation, there are sometimes that new versions must be produced in order to check bugs or even improving some details.

In our case, we have three versions to talk about.

First Version. Basic Controller implements the different methods a controller usually has.

- Creation
- Deletion
- Listing
- Edition

Second Version. List method did not have a requestURI param. This is necessary when a list of items has more than 5 elements (Number predeterminate by configuration). The obtained result when this parameter wasn´t present was “Oops. You have not access”.

Third Version. Form Objects. When this controller was implemented, it was considered for domain entities (T extends DomainEntity). A new version of this controller when making CustomForm (for CustomisationSystem) was delivered immediately.

*For certain use cases, this controller has been modified. For example, when a position has assigned a ticker which exists in the database.

3. Code Explanation

Abstract Methods	

Code	`public abstract <T> ModelAndView saveAction(T e, BindingResult binding, String nameResolver);`
	
	`public abstract <T extends DomainEntity> ModelAndView deleteAction(T e, String nameResolver);`

Parameters	- T e. Entity to work with. - BindingResult binding. When an entity is going to be edited or save this parameter save information about errors. - String nameResolver. This is the view that a user is going to be redirected when a save or delete operation is executed without problems.

List	

Code	`public <T extends DomainEntity> ModelAndView listModelAndView(final String nameCollection, final String nameView, final Collection<T> objects, final String requestURI) {`
	`ModelAndView result;`
	`result = this.custom(new ModelAndView(nameView));`
	`result.addObject(nameCollection, objects);`
	`result.addObject("requestURI", requestURI);`
	`return result;`
	`}`

Parameters	- nameCollection. The name of parameters that any controller will send to a list view. - nameView. List view is going to be used. - Objects. Objects to visualize. - requestURI. Parameter necessary when a list is bigger than 5 objects.

Create	

Code	`public <T extends DomainEntity> ModelAndView create(final T e, final String nameEntity, final String nameView, final String requestURI, final String requestCancel) {`
	`return this.createAndEditModelAndView(e, nameEntity, nameView, requestURI, requestCancel);`
	`}`

Parameters	- e. Object to create or edit. - nameEntity. - nameView. - requestURI. Url to post object when a form is filled. - requestCancel. URI to redirect when user cancels an action.

Show y Edit	Both are implemented on the same way. It is only differenced by a parameter called “view”. View takes true when an object is shown and false in other case.

Code	`public <T extends DomainEntity> ModelAndView edit(final T e, final String nameEntity, final String nameView, final String requestURI, final String requestCancel) {`
	`ModelAndView result;`
	`result = this.createAndEditModelAndView(e, nameEntity, nameView, requestURI, requestCancel);`
	`result.addObject("view", false);`
	`return result;`
	`}`

Parameters	- Same as create

Save	

Code	`public <T extends DomainEntity> ModelAndView save(final T e, final BindingResult binding, final String error, final String nameEntity, final String nameView, final String requestURI, final String requestCancel, final String nameResolver) {`
	
	`ModelAndView result = null;`
	
	`try {`
	`//Llamo al metodo abstracto descrito arriba. En mi clase hija solo tendre que reevaluar el contenido del mismo.`
	`result = this.custom(this.saveAction(e, binding, nameResolver));`
	`} catch (final ValidationException valid) {`
	`result = this.createAndEditModelAndView(e, nameEntity, nameView, requestURI, requestCancel);`
	`} catch (final Throwable oops) {`
	`result = this.createAndEditModelAndView(e, error, nameEntity, nameView, requestURI, requestCancel);`
	`result.addObject("duplicate", true);`
	`}`
	
	`return result;`
	`}`

Sections	- Inside try. We have the abstract operation defined at the top of our class. This method will be override everytime a Controller Class is extended to this. (BasicController). - Catch – ValidationException. When an object is trying to be reconstructed, if any parameter gets failed when validating this exception is thrown on service, making controller takes redirection to the present form. - Catch – Throwable oops. If any exception uncommon is thrown, the user is going to keep on the current view because a certain operation has not been executed properly.

Delete	

Code	`public <T extends DomainEntity> ModelAndView delete(final T e, final String error, final String nameEntity, final String nameView, final String requestURI, final String requestCancel, final String nameResolver) {`
	`ModelAndView result;`
	`try {`
	`result = this.custom(this.deleteAction(e, nameResolver));`
	`} catch (final Throwable oops) {`
	`result = this.createAndEditModelAndView(e, error, nameEntity, nameView, requestURI, requestCancel);`
	`}`
	`return result;`
	`}`

Parameters	- Delete abstract operation is override it the same way as save operation. - Same as save operation with no ValidationException catch. Delete action has not reconstruction, so it´s not necessary.

4. Improvements

For next deliverables, this controller is expected to have removed the entity name parameter. This value will directly taken from entity class name using Reflexive Programming with the following code:

```
String s = e.getClass().getSimpleName();

s = s.replace(s.charAt(0), Character.toLowerCase(s.charAt(0));
```

We must check this previously when a generical object is taken. That´s why we have not implemented it yet. We may have some problems.

Also, we want to check if save and delete operations works ok with “<T> ModelAndView” method (implemented at the beginning for ObjectForm. We want to avoid duplicate code as possible.