

Welcome to this lesson! Our goal's to keep putting the theory we've learnt in the previous lectures into practice.

--

Copyright (C) 2017 Universidad de Sevilla

The use of these slides is hereby constrained to the conditions of the TDG Licence, a copy of which you may download from <http://www.tdg-seville.info/License.html>

The problem



UNIVERSIDAD DE SEVILLA

Today's problem's a little more involved than the previous one: an event-manager that will help organisers organise their events and customers register to them.

Ready to keep riding?



UNIVERSIDAD DE SEVILLA

Please, note that this project's a little more involved, but we still provide a lot of support. It's like riding this bike because we provide a project in which you only have to create a few views for it to work.

You're constrained, yet!



UNIVERSIDAD DE SEVILLA

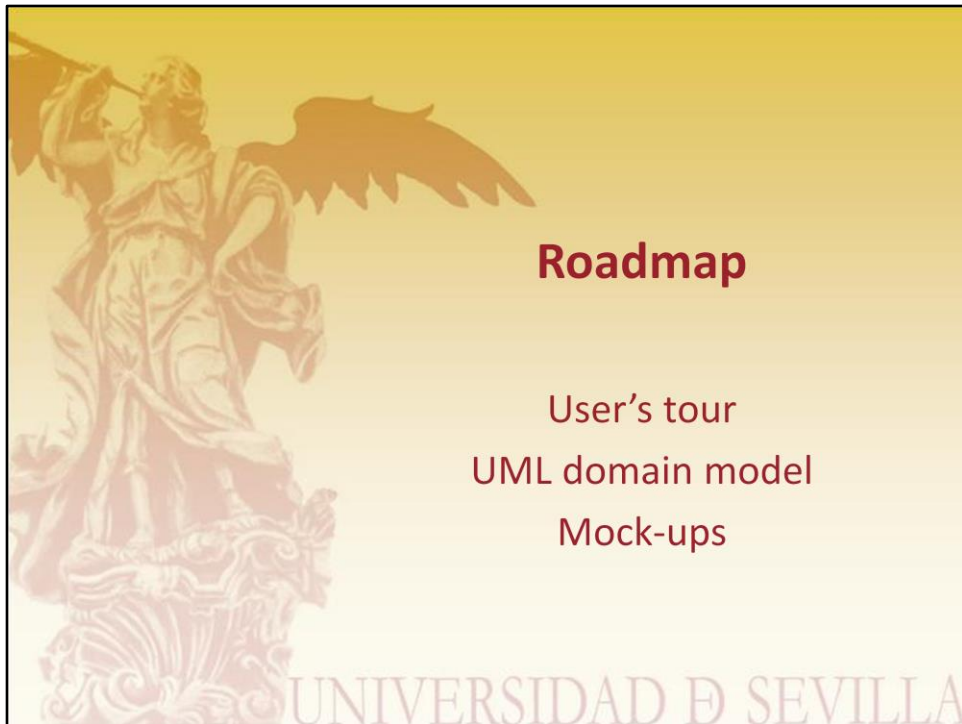
Please, note that you're totally constrained. Your only duty in this project is to implement some views; you aren't allowed to change anything else. Please, take this very seriously: the other parts of the project were implemented by other people; you can't change the artefacts they've produced and hope this doesn't mess things up!

Remember: your first work day

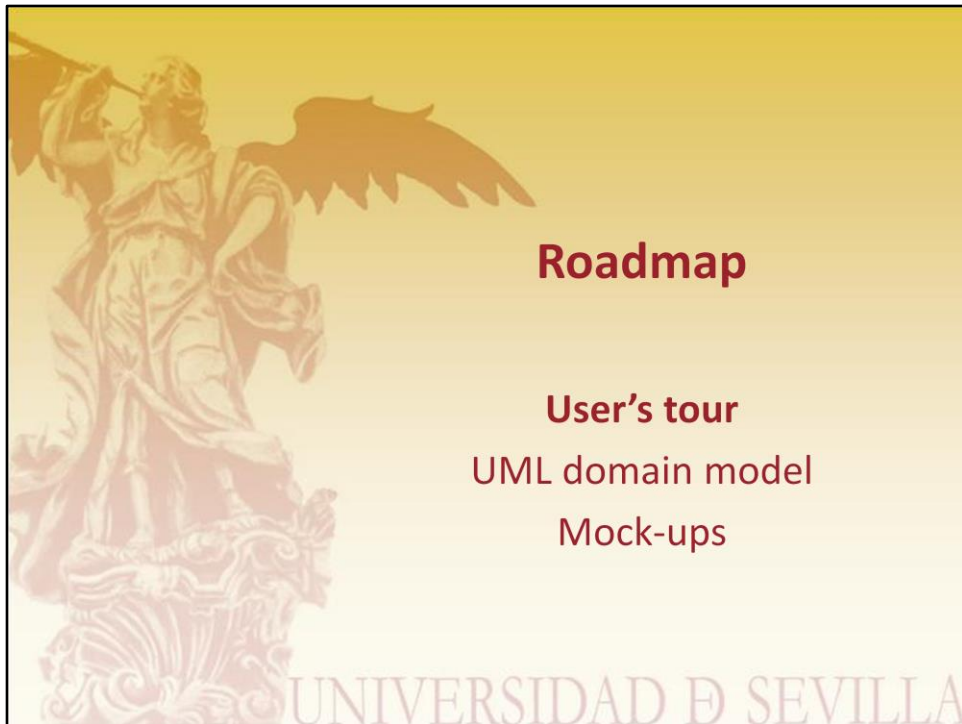
Hi ho! Hi ho! Hi ho! Hi ho!
It's to our first work day we go!



Please, realise that this problem's very similar to the problems that you'll have to solve on your first day as a professional software engineer: your boss will assign you a very simple task like creating some views building on a specification that will be very similar to ours.



This is our roadmap. It won't be too long since the goal's that you can start working on your project as soon as possible. We'll start with a user's tour; then we'll present the UML domain model, and we'll conclude with a description of the mock-ups that you have to implement.



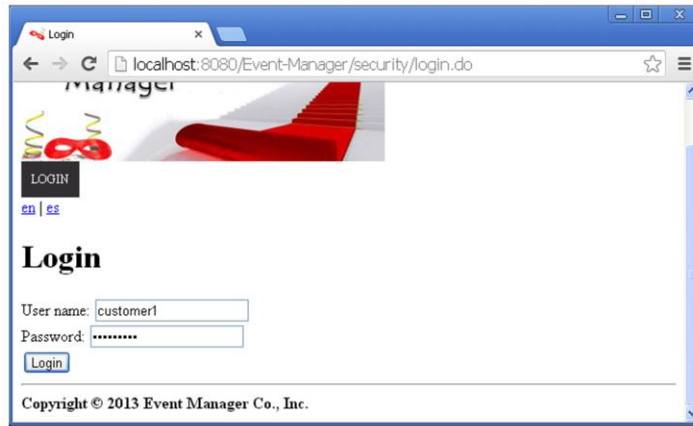
Let's start with the user's tour. In this section, our goal's to present a number of screenshots so that you can have a better understanding of what we expect from this project.

The welcome screen



This is the welcome screen. By now, you should be very familiar with it. It has a logo, a simple menu that just displays “Login”, a language bar, and a welcome message.

Logging in as a customer



You may log in as a customer using two pre-defined customer accounts, namely: "customer1/customer1" and "customer2/customer2".

The main menu



Once you're logged in as a customer, the menu will change and will show a couple of options, namely: "My events" and "New events".

My events

en | es

List of events

3 items found, displaying all items.

1

	Title	Moment	Description	Price
Unregister	Event 1	12/12/2012 12:12	Blah, blah, blah	100.0
Unregister	Event 2	12/12/2013 12:12	Blah, blah, blah	12.34
Unregister	Event 3	12/12/2012 12:12	Blah, blah, blah	100.0

Copyright © 2013 Event Manager Co., Inc.

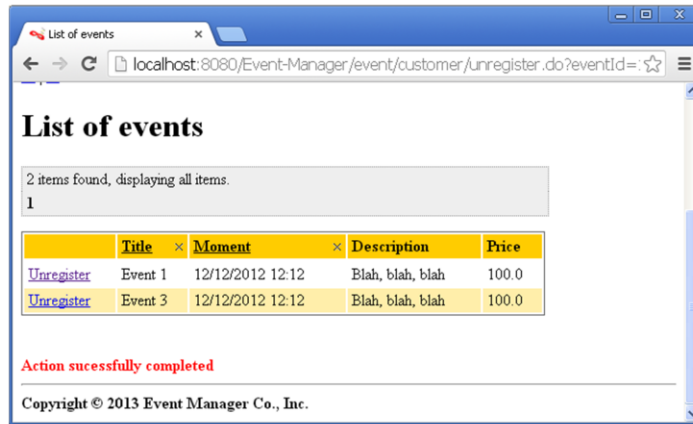
This is what the “My events” option shows to a customer: a list with the events to which he or she’s registered previously. Note that, as usual, the listing may be sorted according to some of the attributes, and that there’s a navigation pane on top of the listing. In this case, there’s not an “Edit” link or a “Create event” link since a customer cannot edit or create events; note, however, that there’s an “Unregister” link that allows a customer to unregister from an event as long as it hasn’t been organised, yet.

Unsuccessful unregistration



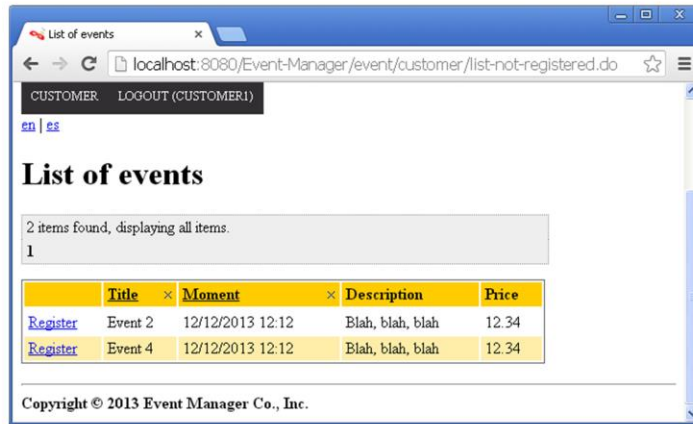
The first and the third events have already been organised, so the customer can't unregister from them. If the user clicked on the "Unregister" link of the first event, for instance, he or she would get the error that's shown below the grid.

Successful unregistration



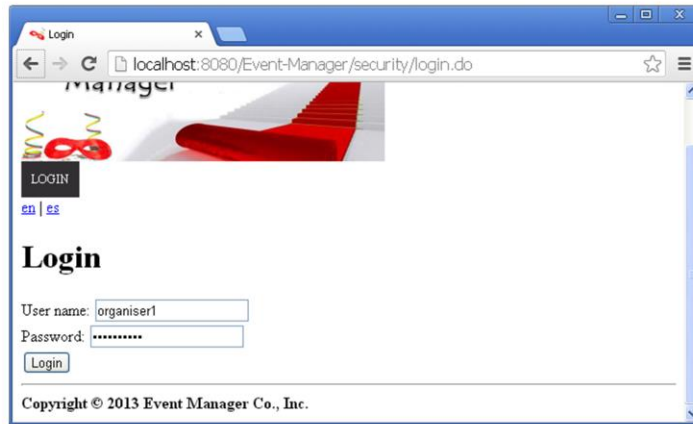
Contrarily, if he or she unregisters from the second event, which hasn't been organised, yet, he or she would get an acknowledge message and the list would be refreshed, as shown in this slide.

New events



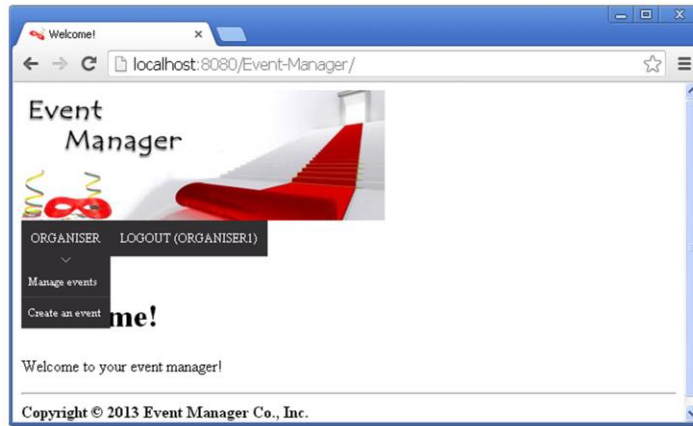
The "New events" option shows a listing with the events in which the customer who's using the application hasn't registered, yet. Note that the first column shows a "Register" link. If the user clicks on it, then the system registers him or her to the event. As was the case before, errors or acknowledge messages are shown on the screen whenever appropriate.

Logging as an organiser



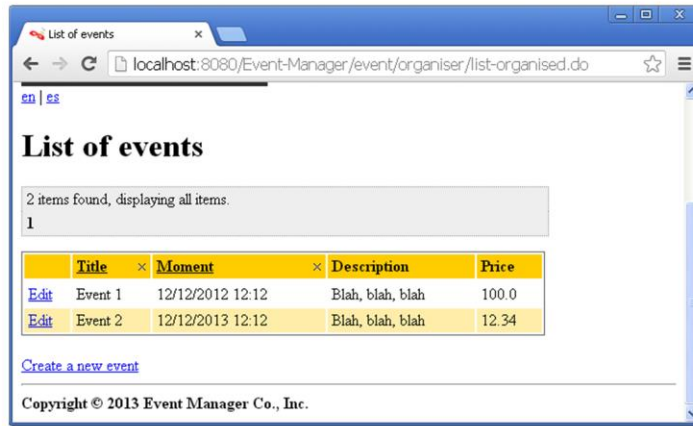
You may log in as an organiser using two pre-defined organiser accounts, namely: “organiser1/organiser1” and “organiser2/organiser2”.

The main menu



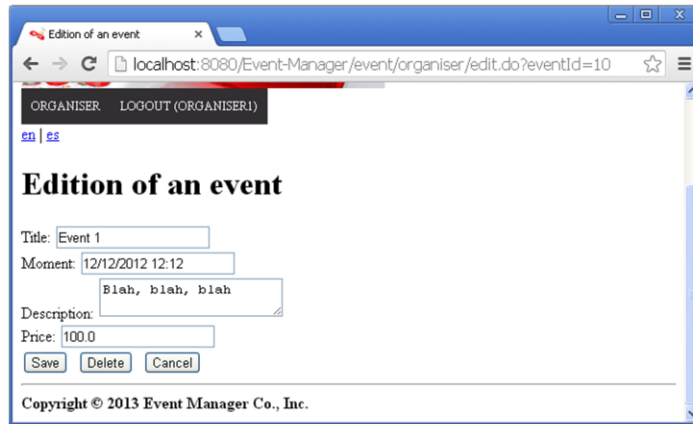
Once you're logged in as an organiser, the menu will change and will show a couple of options, namely: "Manage events" and "Create an event".

Managing events



The option "Manage events" shows a listing with the events that the principal's created. Note that, as usual, there's an "Edit" and a "Create new event" link so that an organiser can change the information related to the events that he or she organises and create new ones.

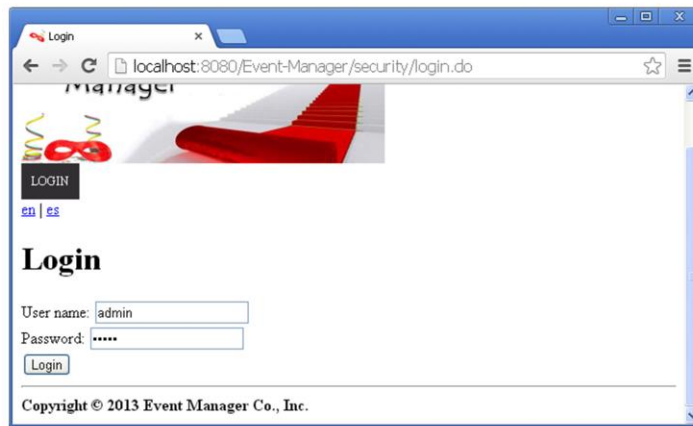
Creating/editing events



The screenshot shows a web browser window with the title 'Edition of an event'. The address bar displays 'localhost:8080/Event-Manager/event/organiser/edit.do?eventId=10'. The page has a dark header with 'ORGANISER' and 'LOGOUT (ORGANISER1)' links. Below the header, there are links for 'en' and 'es'. The main content area is titled 'Edition of an event' and contains a form with the following fields: 'Title' (containing 'Event 1'), 'Moment' (containing '12/12/2012 12:12'), 'Description' (containing 'Blah, blah, blah'), and 'Price' (containing '100.0'). At the bottom of the form are three buttons: 'Save', 'Delete', and 'Cancel'. The footer of the page reads 'Copyright © 2013 Event Manager Co., Inc.'.

This is the screen in which an organiser can create and/or edit the events that he or she organises. We won't provide any other details since this is a typical edition form, like the many we've already seen.

Logging as an administrator

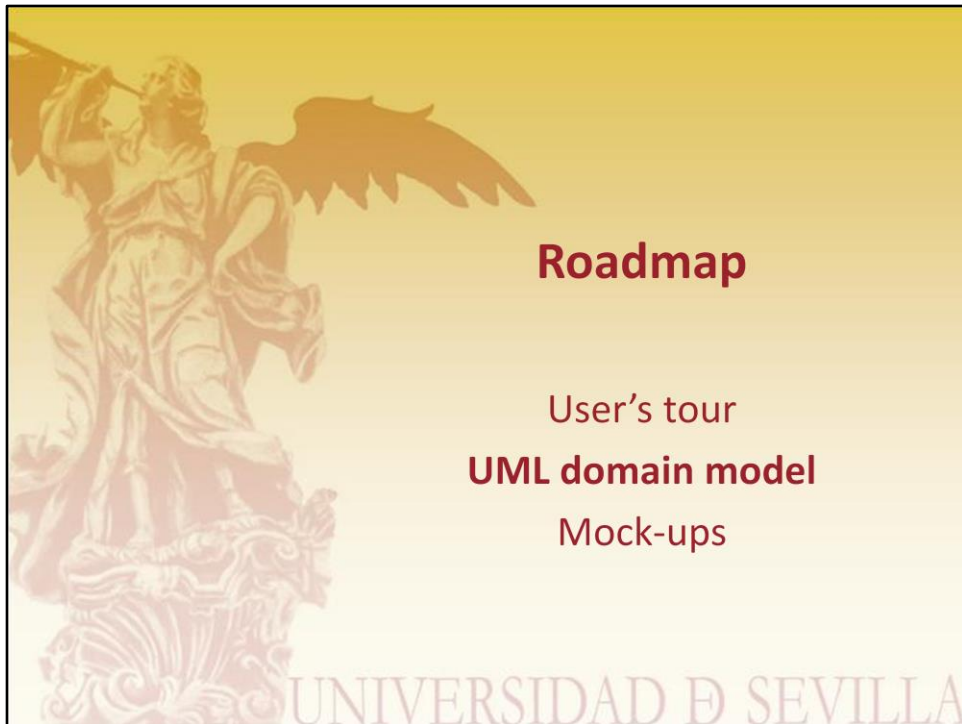


If you wish, you may also log in as an administrator using the “admin/admin” account.

What an admin can do

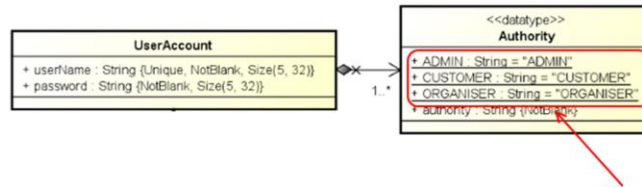


Unfortunately, there's nothing interesting an administrator can do, except for logging out or changing the language in which the application works. We're not interested in the administrator's functionality in this project.



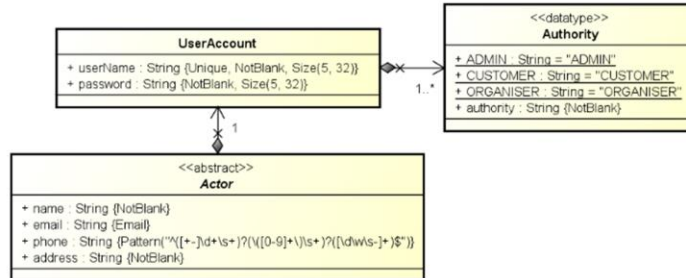
Let's now present the UML domain model.

The model (I)



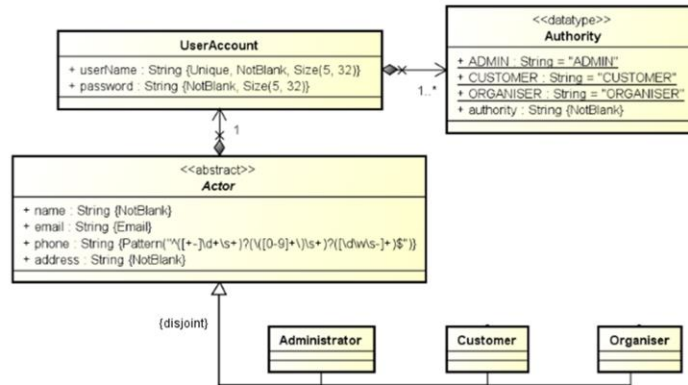
As usual, we'll start with the classes that model user accounts and authorities. Note that you don't have to make any changes to the "UserAccount" class, but need to perform a small change to class "Authority". The reason is that this project deals with an additional authority that isn't provided by our project template: organisers. You just need to add one more authority to this class; that shouldn't be difficult at all since you've already done it regarding your subject project... didn't you?

The model (II)



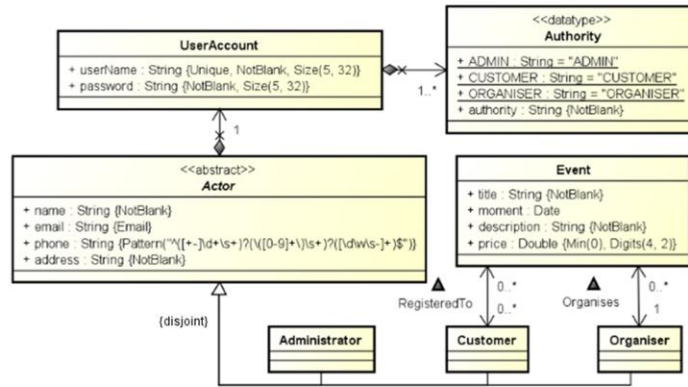
Then, we model class “Actor”, which has the data we store about the actors of the system. As usual, we’ll store their names, emails, phone numbers, and addresses.

The model (III)

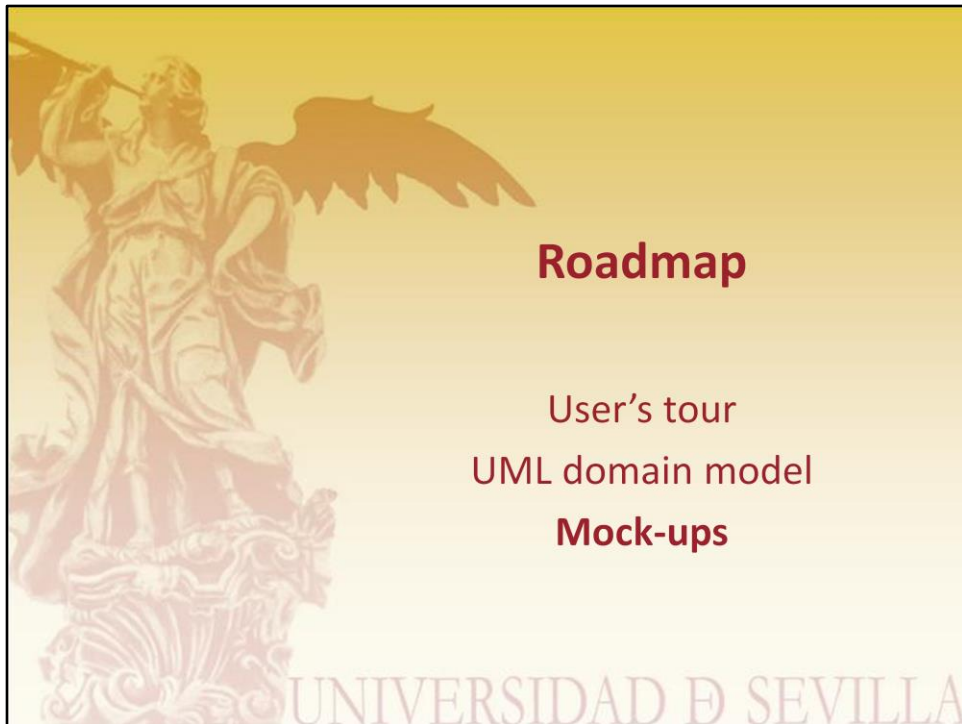


And we specialise class “Actor” into three subclasses, namely: “Administrator”, “Customer”, and “Organiser”, which are disjoint.

The model (IV)

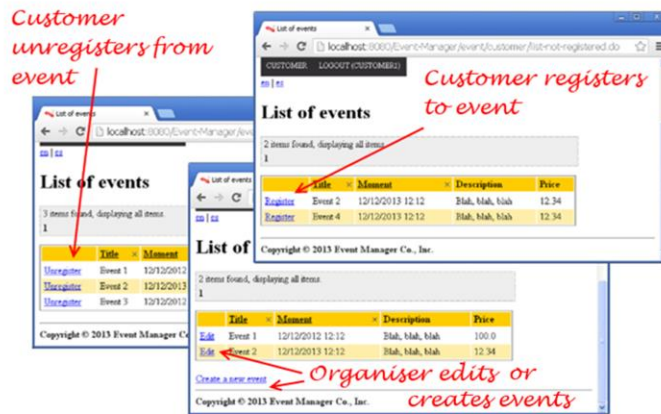


Finally, we introduce a class to model the events. It has attributes to represent their titles, the moments when they're organised, their description, and the price of their tickets. Note that this class has two relationships, namely: "RegisteredTo", to model that a customer may have registered to an arbitrary number of events, and that an event may have an arbitrary number of customers who have registered to it; and "Organises", to model that an organiser organises an arbitrary number of events, and that each event must be organised by exactly one organiser.



The UML domain model was simple, wasn't it? Let's now provide a little more information about the use cases.

Very similar views, aren't they?



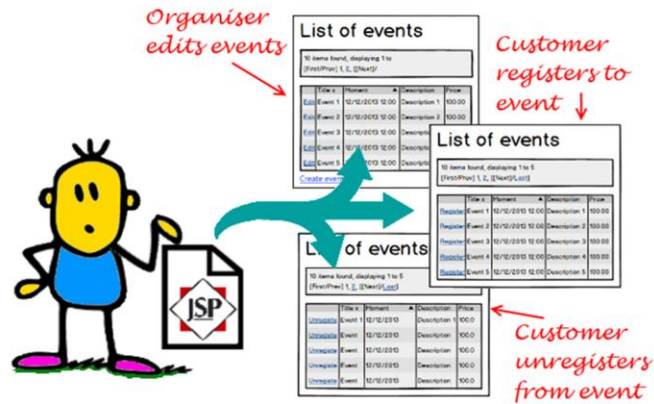
Did you realise that some of the user interfaces in this application are very similar to each other, didn't you? Please, take a look at these screenshots. Can you spot the differences? The body of the listing's exactly the same in every case; the only differences are regarding the first column, which changes depending on the principal and the option that he or she's selected, and a link at the bottom that appears only when the principal's an organiser.

Shall we clone the views?



How shall we implement these views? That's a good question! Shall we clone the views? Obviously, we won't copy and paste a template. That is: we won't create three independent views because this may very easily lead to problems.

One view to rule'em all!



What we're going to do is to create a single JSP document that will result into three different views depending on the principal and the option he or she's selected from the main menu.

This is the actual view

The screenshot shows a web page titled "List of events". It includes a pagination bar with "displaying 1 to 5" and "[Next]/". Below this is a table with columns: "Title x", "Moment", "Description", and "Price". The table contains five rows of event data. At the bottom of the table is a link "Create a new event". Three yellow callout boxes with red arrows point to specific elements:

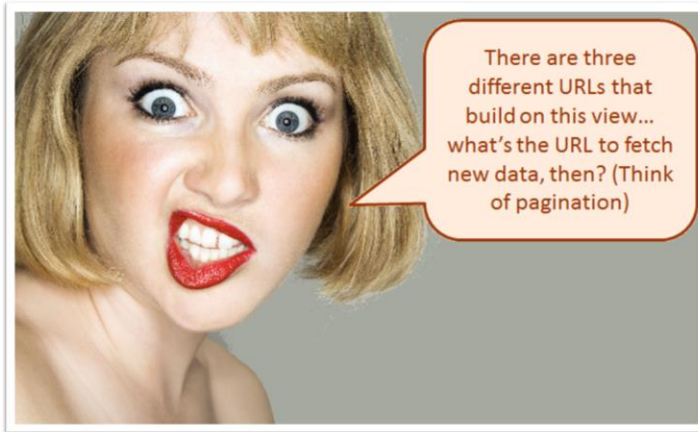
- Box 1 (top left): "Show if the principal's an organiser" points to the "Edit" link in the first row.
- Box 2 (top right): "Show if he or she's a customer who is listing his or her events" points to the "Unregister" link in the first row.
- Box 3 (bottom): "Show if he or she's a customer and is listing new events" points to the "Create a new event" link.

Title x	Moment	Description	Price
Event 1	12/12/2013 12:00	Description 1	100.00
Event 2	12/12/2013 12:00	Description 2	100.00
Event 3	12/12/2013 12:00	Description 3	100.00
Event 4	12/12/2013 12:00	Description 4	100.00
Event 5	12/12/2013 12:00	Description 5	100.00

This is the actual view that we're going to implement. We'll write the code that is necessary to output the three columns and the link at the bottom, but we'll use the "authorize" tag that is provided by the Spring security tag library in order to prevent a principal from seeing the links that aren't available to him or her. Please, review the theory lecture in this lesson to learn how to use this tag.

WARNING: recall that hiding a link doesn't mean that the system's protected. The user can make his or her browser for arbitrary URLs, including URLs that aren't allowed to him or her. So far, we haven't worked a lot on this problem. We'll study this in the next lesson, the one that we devote to controllers.

Wait a minute!



Wait a minute! We've told you that there are three different actions that build on this view. In other words, that there are three different URLs that request this view. It's a listing view, so we need to write something like the following tag:

```
<display:table name="events" id="row" pagesize="5" class="displaytag"
    requestURI="{your-page's-url}" >
...
</display:table>
```

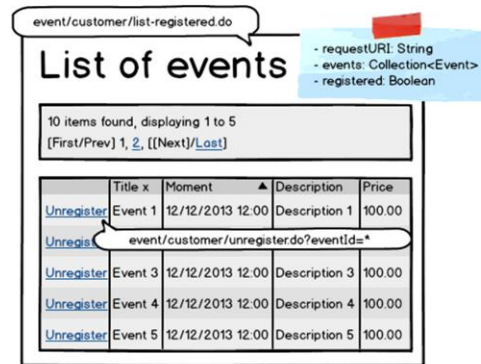
The problem should be clear to you now: which URL should we use to initialise the "requestURI" parameter? It depends on the use case, so we can't write it verbatim in the view.

Quite a simple solution



Fortunately, the answer to the previous question is very easy: use a model variable to indicate the URL. We hope you understand the idea. Please, don't go ahead unless you understood it.

My events (customer)



This is the mock-up that for the use case in which a customer lists the events in which he or she's registered. There's little to say regarding the design of the URLs; regarding the model, we have a couple of notes:

- Note that the model includes a variable called "requestURI", which is a string whose value is expected to be the URL of the page in which this view is shown. Please, recall that the problem with this project is that we must design a view that serves several use cases. So we need to include a variable in the model that indicates the URL to which the listing has to make additional requests if pagination is necessary. In this case, variable "requestURI" must be set to "event/customer/list-registered.do". Was it clear to you? Please, don't go ahead unless you've understood this idea.
- The collection of events to be displayed is stored in a model variable called "events", this is quite straightforward.
- There's an additional Boolean variable that indicates whether the "Unregister" or the "Register" links must be shown. Since we use the same view to display both the events to which a customer has registered and the events to which he or she hasn't registered, yet, this Boolean variable indicates which of the cases the view's dealing with. To produce this user interface, variable "registered" must be set to "true".

New events (customer)

event/customer/list-unregistered.do

List of events

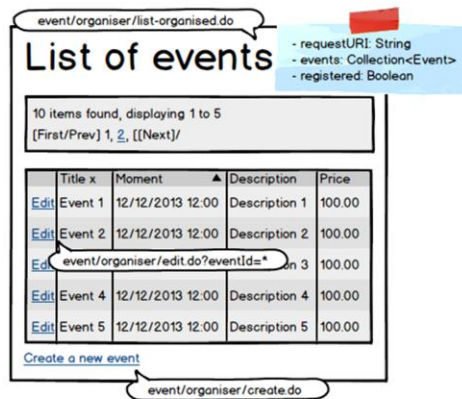
- requestURI: String
- events: Collection<Event>
- registered: Boolean

10 items found, displaying 1 to 5
[First/Prev] 1, 2, [[Next]/Last]

	Title x	Moment	Description	Price
Register	Event 1	12/12/2013 12:00	Description 1	100.00
Register	Event 2	12/12/2013 12:00	Description 2	100.00
Register	event/customer/register.do?eventId=*			100.00
Register	Event 4	12/12/2013 12:00	Description 4	100.00
Register	Event 5	12/12/2013 12:00	Description 5	100.00

This is the mock-up to show the events in which a customer isn't registered, yet. It's very similar to the previous one, the only difference being that instead of "Unregister" links, the mock-up shows "Register" links. Obviously, this view must be fed with a model in which variable "requestURI" is set to "event/customer/list-unregistered.do" and variable "registered" is set to "false".

Manage events (organiser)



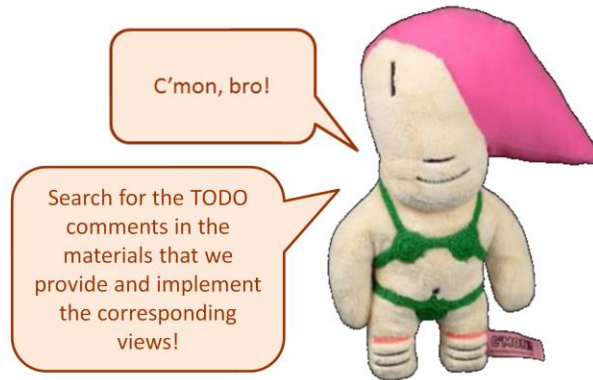
This is the mock-up that provides a user interface to the following use case: an organiser lists the events that he or she's organised. It's very similar to the previous ones, the only difference being that instead of "Unregister" or "Register" links, the mock-up shows "Edit" links and there's an additional "Create a new event" link below the grid. In this case, variable "registered" does not make sense, so it's value can be "null".

Edit event (organiser)



This is the mock-up that provides a user interface to the use case in which an organiser edits one of the events that he or she's organising. You should be able to interpret it very easily, now.

Ready to start working?



Ready to start working? C'mon!



Thanks for attending this lecture! See you next day!