Step by Step Guide to the Bicycle Workshop Application

As Learning by Doing is the most effective way to get started with a new framework, this document will guide you through the key features of the CUBA Platform framework and show how you can accelerate development of enterprise applications, taking bicycle workshop system as an example.

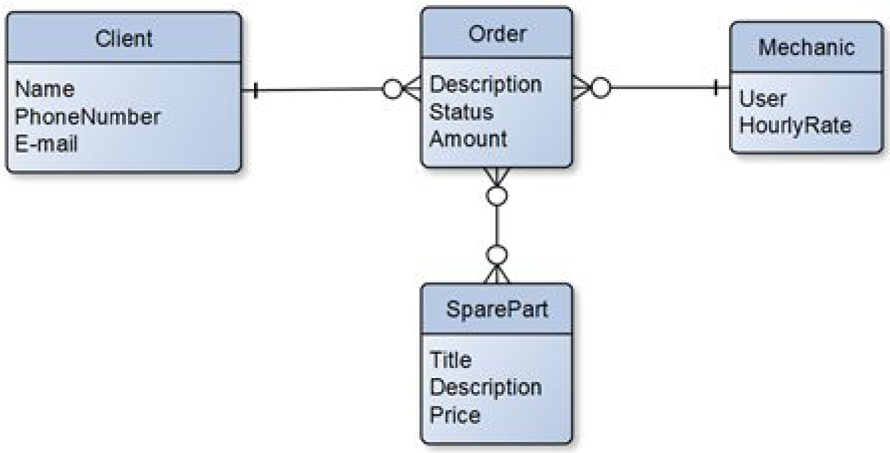
Estimated time to complete this lab is 2 hours[1].

1 FUNCTIONAL SPECIFICATION

The application should meet the following requirements:

1. Store customers with their name, mobile phone and email
2. Record information about orders: price for repair and time spent by mechanic
3. Keep track of spare parts in stock
4. Automatically calculate price based on spare parts used and time elapsed
5. Control security permissions for screens, CRUD operations and records' attributes
6. Audit of critical data changes
7. API for a mobile client to place an order

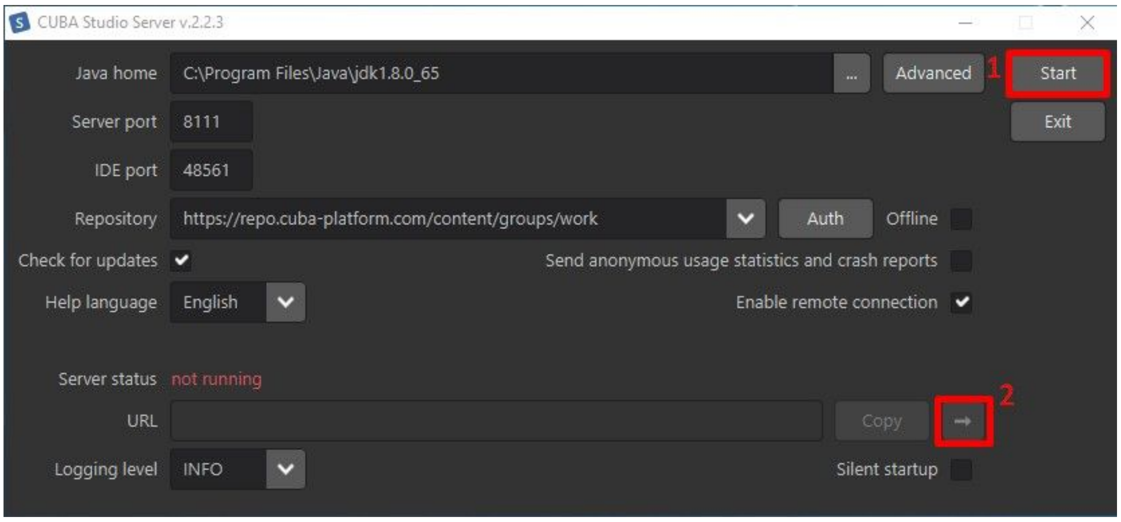
Data model is shown in the picture below:



[1] The estimation is given for developers, who have general (basic) knowledge of Java SE.

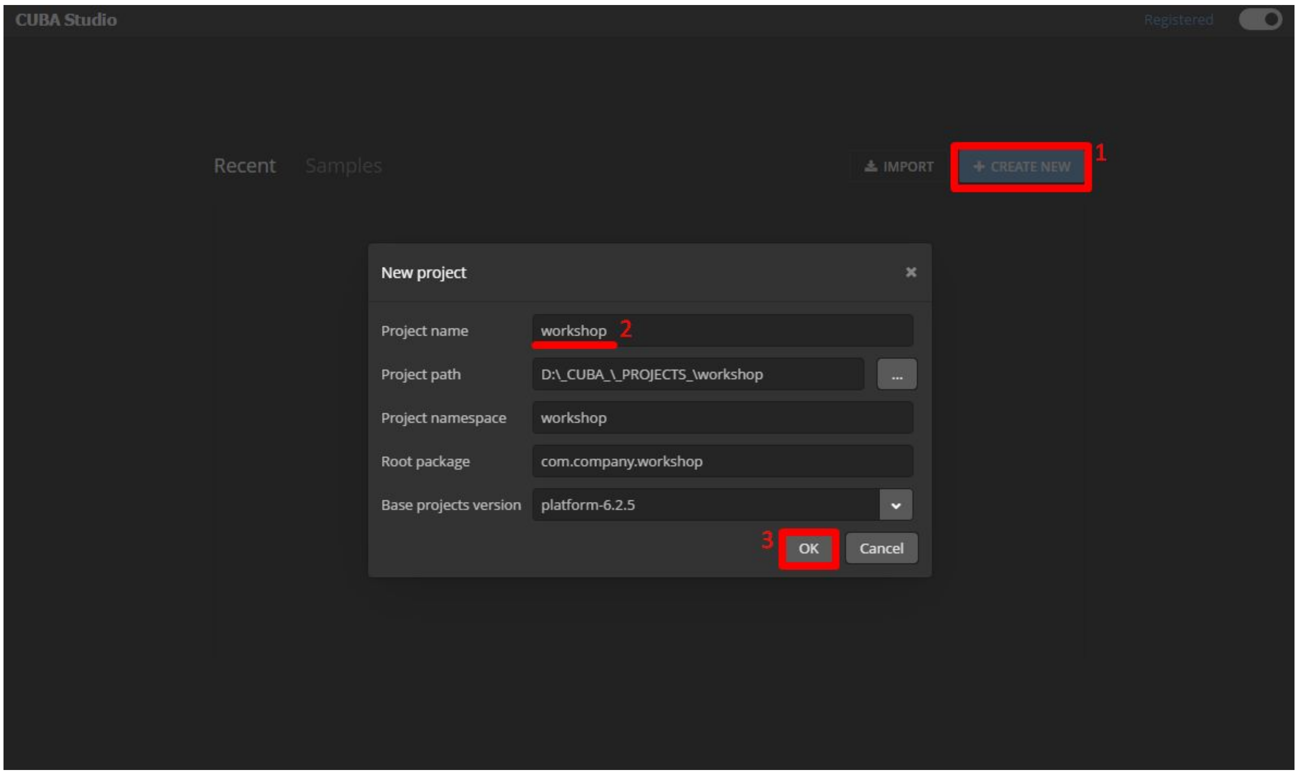
2 CREATING A NEW PROJECT

1. Start the CUBA Studio server and open the Studio in your browser as it is shown in the picture below:


CUBA Studio Server v.2.2.3

Java home: C:\Program Files\Java\jdk1.8.0_65
Server port: 8111
IDE port: 48561
Repository: https://repo.cuba-platform.com/content/groups/work
Check for updates: ✓
Send anonymous usage statistics and crash reports: ☐
Help language: English
Enable remote connection: ✓
Server status: not running
URL:
Logging level: INFO
Silent startup: ☐

2. After completing the previous step you can find the CUBA Studio up and running in your browser on the http://localhost:8111/studio/ URL. To create a new project:
   2.1. Click **Create New** on welcome screen
   2.2. Fill up the **Project name** field: ***workshop***
   2.3. Complete the step by clicking **OK**

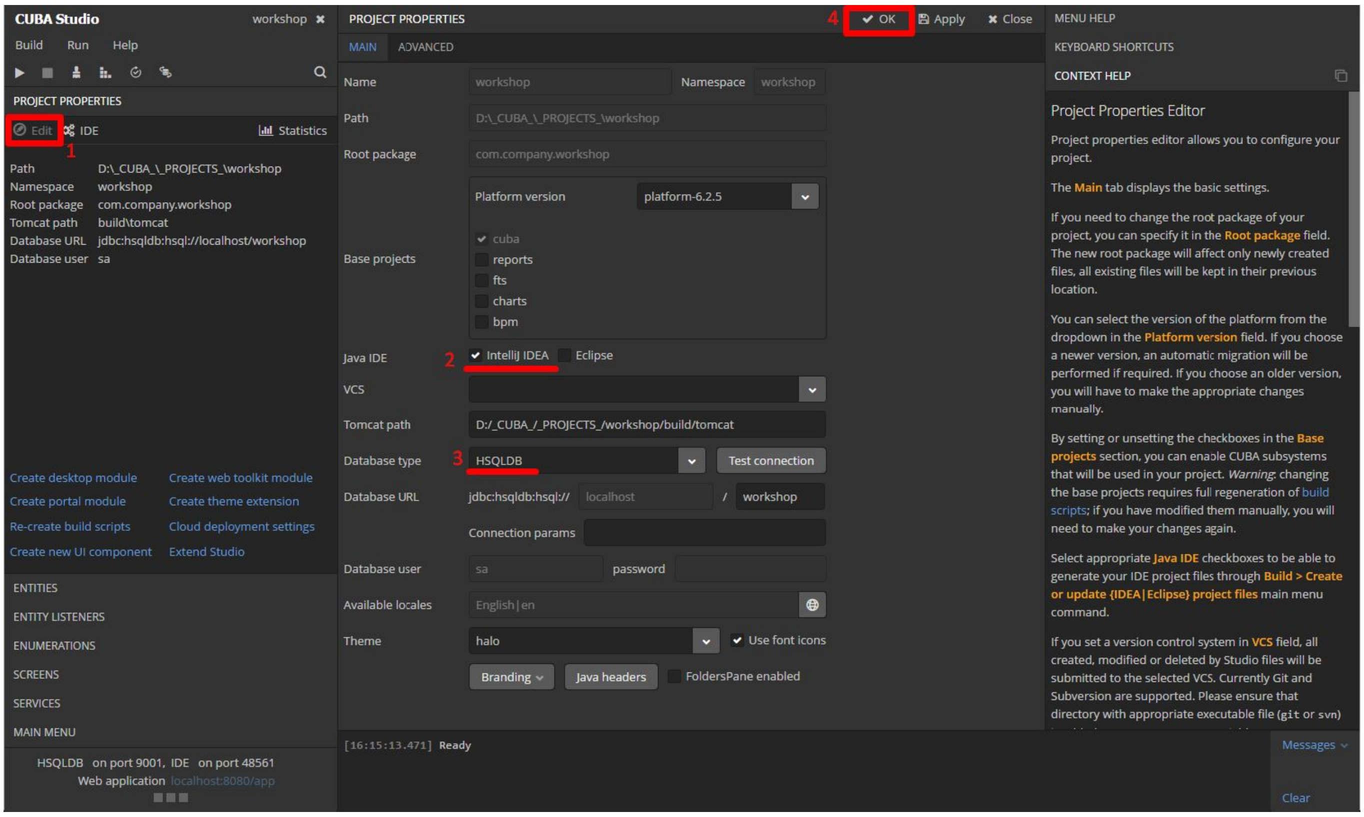

New project

Project name: workshop
Project path: D:_CUBA__PROJECTS_\workshop
Project namespace: workshop
Root package: com.company.workshop
Base projects version: platform-6.2.5

3. Now we will set up global properties of the project. Press **Edit** on the **Project Properties** panel and check the following parameters:
   3.1. **Java IDE**: ***IntelliJ IDEA*** to be checked

3.2. **Database type**: ***HSQL*** to be selected

3.3. Press **OK**



Path D:_CUBA__PROJECTS_\workshop
Namespace workshop
Root package com.company.workshop
Tomcat path build\tomcat
Database URL jdbc:hsqldb:hsql://localhost/workshop
Database user sa

HSQLDB on port 9001, IDE on port 48561
Web application localhost:8080/app

Name: workshop — Namespace: workshop
Path: D:_CUBA__PROJECTS_\workshop
Root package: com.company.workshop
Platform version: platform-6.2.5
Base projects: [✓ cuba] reports fts charts bpm
Java IDE: [✓ IntelliJ IDEA] Eclipse
VCS:
Tomcat path: D:/_CUBA_/_PROJECTS_/workshop/build/tomcat
Database type: HSQLDB
Database URL: jdbc:hsqldb:hsql:// localhost / workshop
Connection params:
Database user: sa — password:
Available locales: English|en
Theme: halo — [✓ Use font icons]
FoldersPane enabled

```
[16:15:13.471] Ready
```

Project Properties Editor

Project properties editor allows you to configure your project.

The **Main** tab displays the basic settings.

If you need to change the root package of your project, you can specify it in the **Root package** field. The new root package will affect only newly created files, all existing files will be kept in their previous location.

You can select the version of the platform from the dropdown in the **Platform version** field. If you choose a newer version, an automatic migration will be performed if required. If you choose an older version, you will have to make the appropriate changes manually.

By setting or unsetting the checkboxes in the **Base projects** section, you can enable CUBA subsystems that will be used in your project. *Warning*: changing the base projects requires full regeneration of build scripts; if you have modified them manually, you will need to make your changes again.

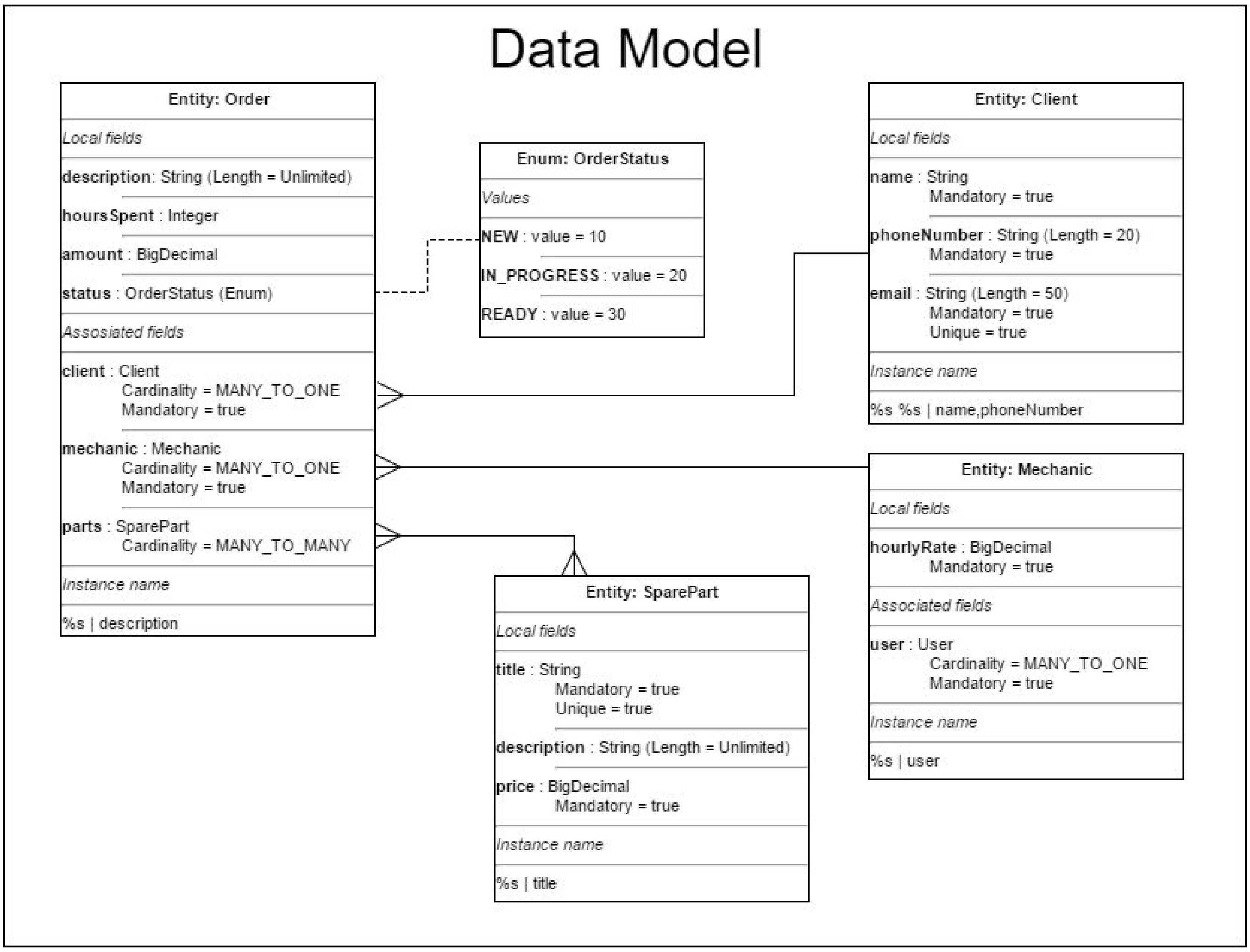
Select appropriate **Java IDE** checkboxes to be able to generate your IDE project files through **Build > Create or update {IDEA|Eclipse} project files** main menu command.

If you set a version control system in **VCS** field, all created, modified or deleted by Studio files will be submitted to the selected VCS. Currently Git and Subversion are supported. Please ensure that directory with appropriate executable file (`git` or `svn`)

Now our project is properly configured and everything is ready to start working on the data model.

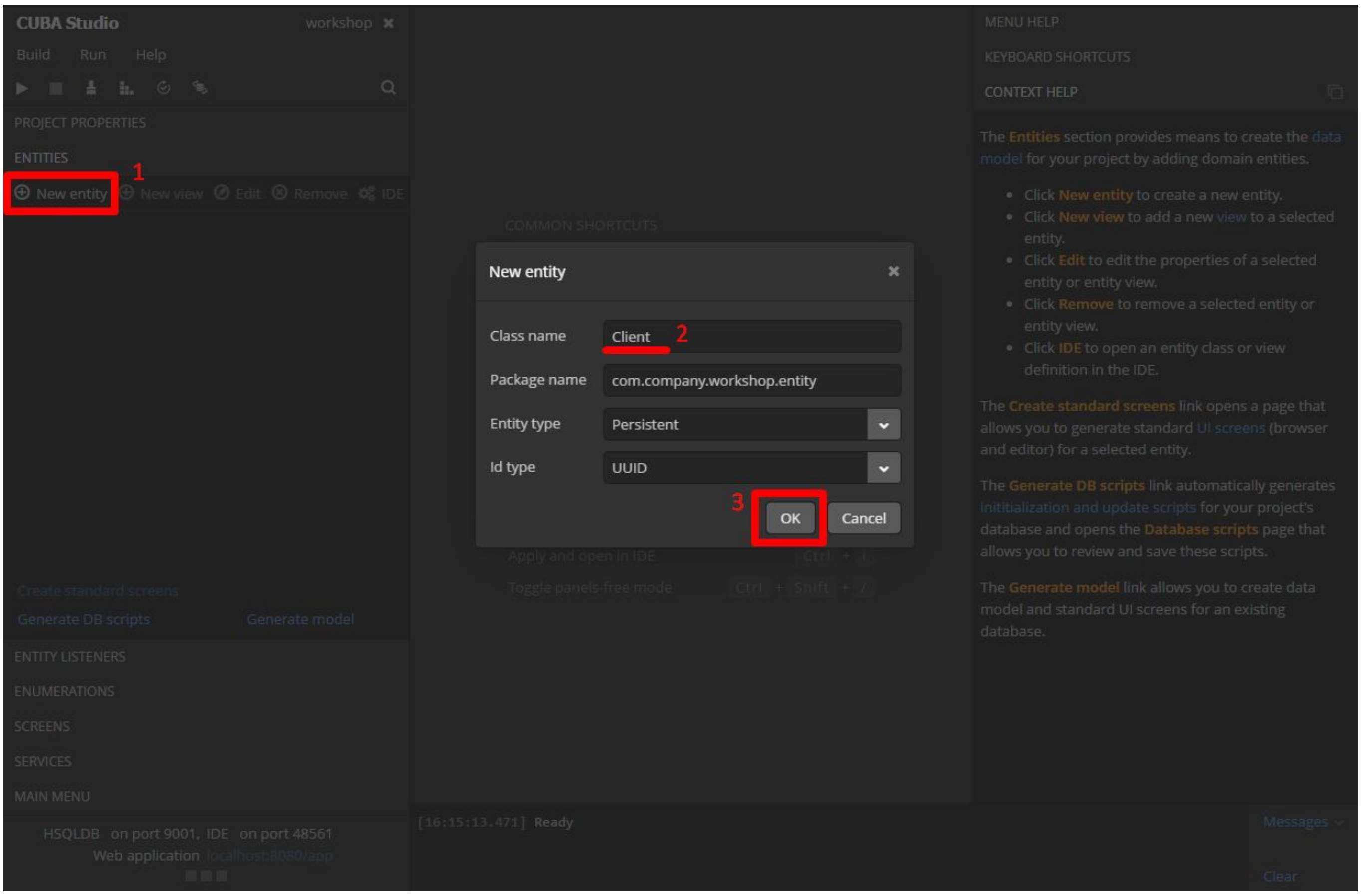
3 DEFINING DATA MODEL AND CREATING THE DATABASE

According to the ER diagram we will create 4 entities: ***Client***, ***Mechanic***, ***SparePart*** and ***Order*** and one enumeration ***OrderStatus***.



3.1 Client Entity

1. Open the **Entities** section of the left hand side navigation panel and click **New entity**
2. Set **Class name**: ***Client***
3. Click **OK**



New entity

Class name: Client
Package name: com.company.workshop.entity
Entity type: Persistent
Id type: UUID

```
[16:15:13.471] Ready
```

The Entities section provides means to create the data model for your project by adding domain entities.

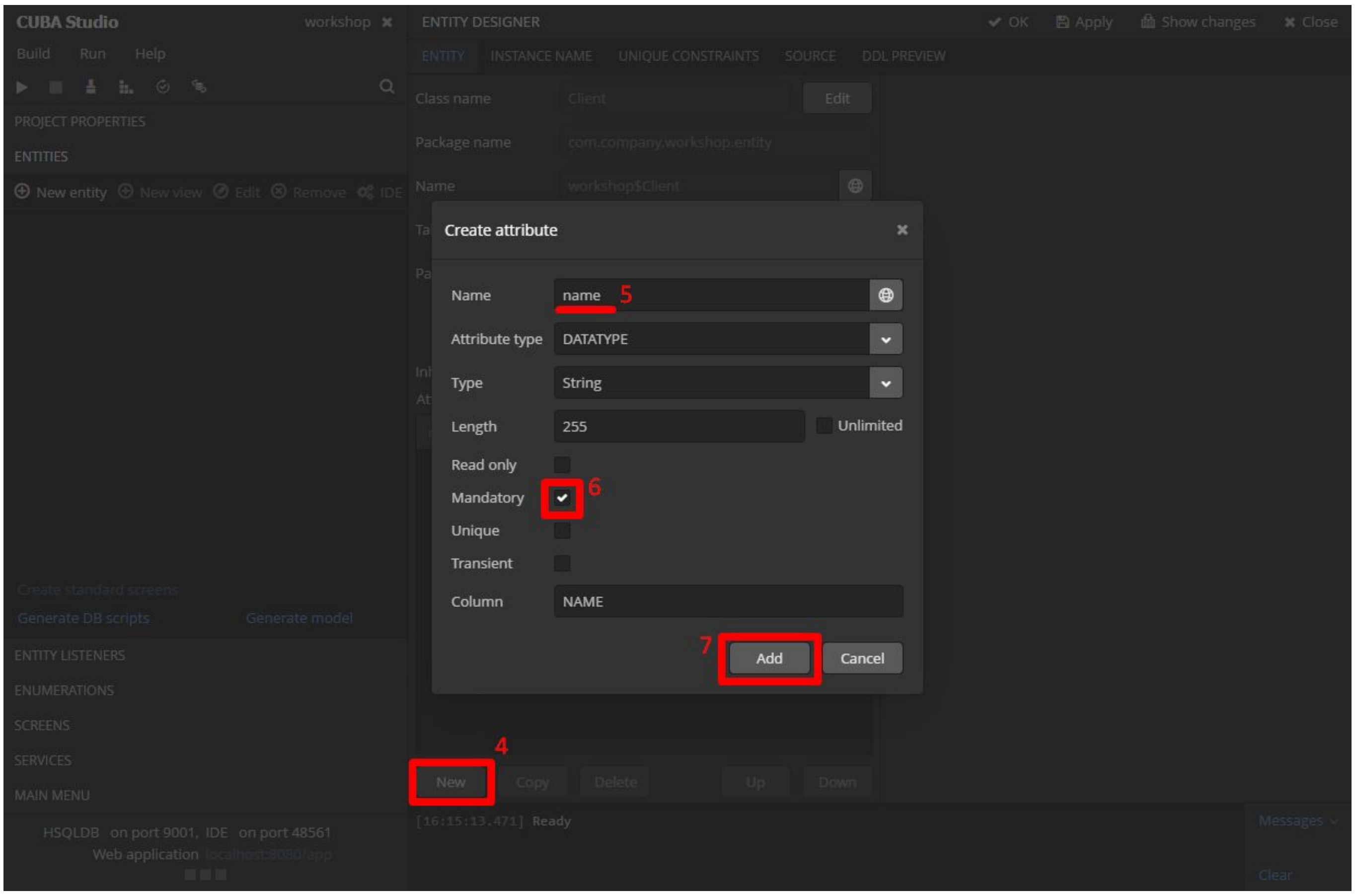
- Click New entity to create a new entity.
- Click New view to add a new view to a selected entity.
- Click Edit to edit the properties of a selected entity or entity view.
- Click Remove to remove a selected entity or entity view.
- Click IDE to open an entity class or view definition in the IDE.

The Create standard screens link opens a page that allows you to generate standard UI screens (browser and editor) for a selected entity.

The Generate DB scripts link automatically generates initialization and update scripts for your project's database and opens the Database scripts page that allows you to review and save these scripts.

The Generate model link allows you to create data model and standard UI screens for an existing database.

4. Click the **New** button under the **Attributes** table
5. Enter **Name:** ***name***
6. Select the **Mandatory** checkbox
7. Click **Add**



Class name: Client
Package name: com.company.workshop.entity
Name: workshop$Client

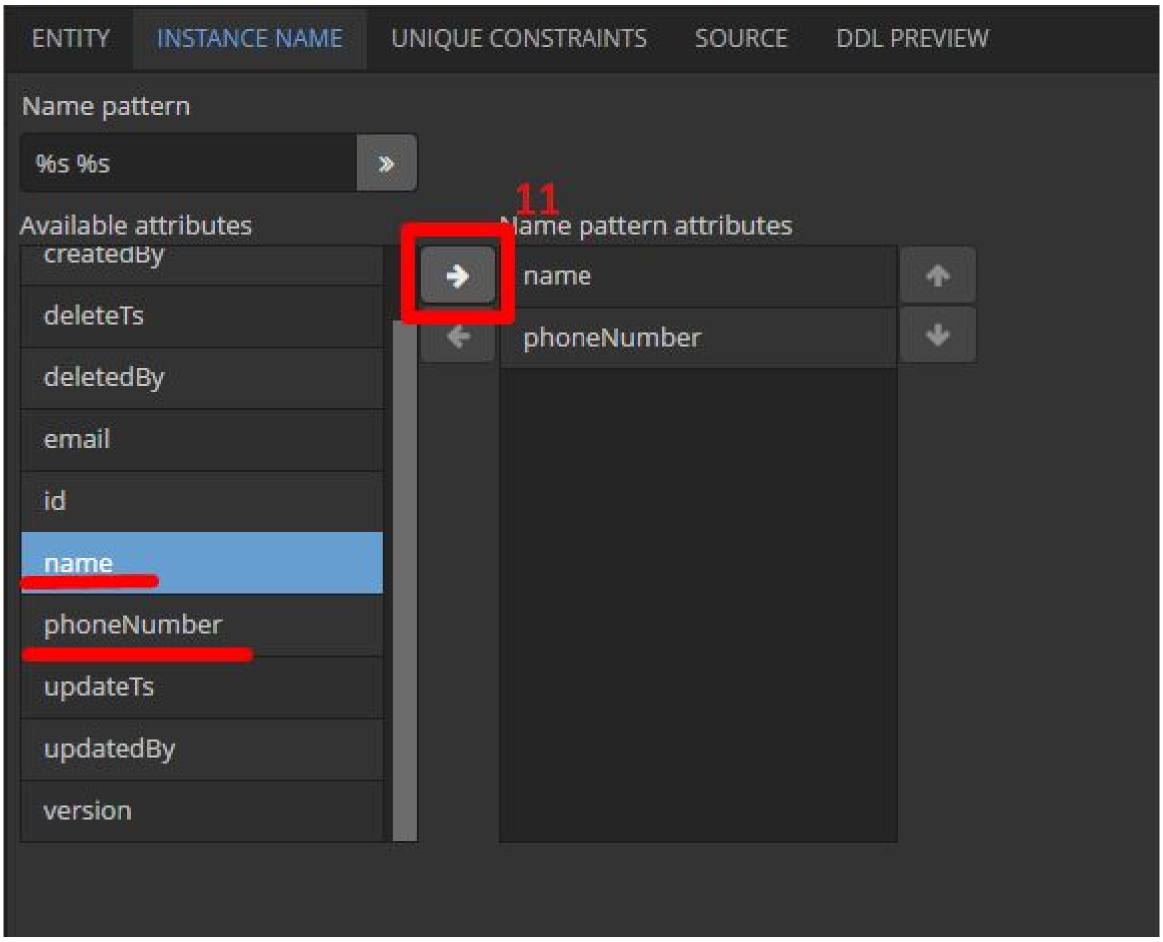
Create attribute

Name	name
Attribute type	DATATYPE
Type	String
Length	255 [ ] Unlimited
Read only	[ ]
Mandatory	[✓]
Unique	[ ]
Transient	[ ]
Column	NAME

HSQLDB on port 9001, IDE on port 48561
Web application localhost:8080/app

```
[16:15:13.471] Ready
```

8. Repeat steps 4-7 to create ***phoneNumber*** as a **mandatory string** attribute with the **length of 20** and flagged as **unique**
9. Repeat steps 4-7 to create ***email*** as a **mandatory string** attribute with the **length of 50** and flagged as **unique**
10. Go to the **Instance name** tab of the entity designer screen. On this tab you can specify the default string representation of an entity to be shown in tables, dropdowns, etc.
11. Select ***name*** and ***phoneNumber*** fields one by one



Name pattern: %s %s

Available attributes: createdBy, deleteTs, deletedBy, email, id, name, phoneNumber, updateTs, updatedBy, version

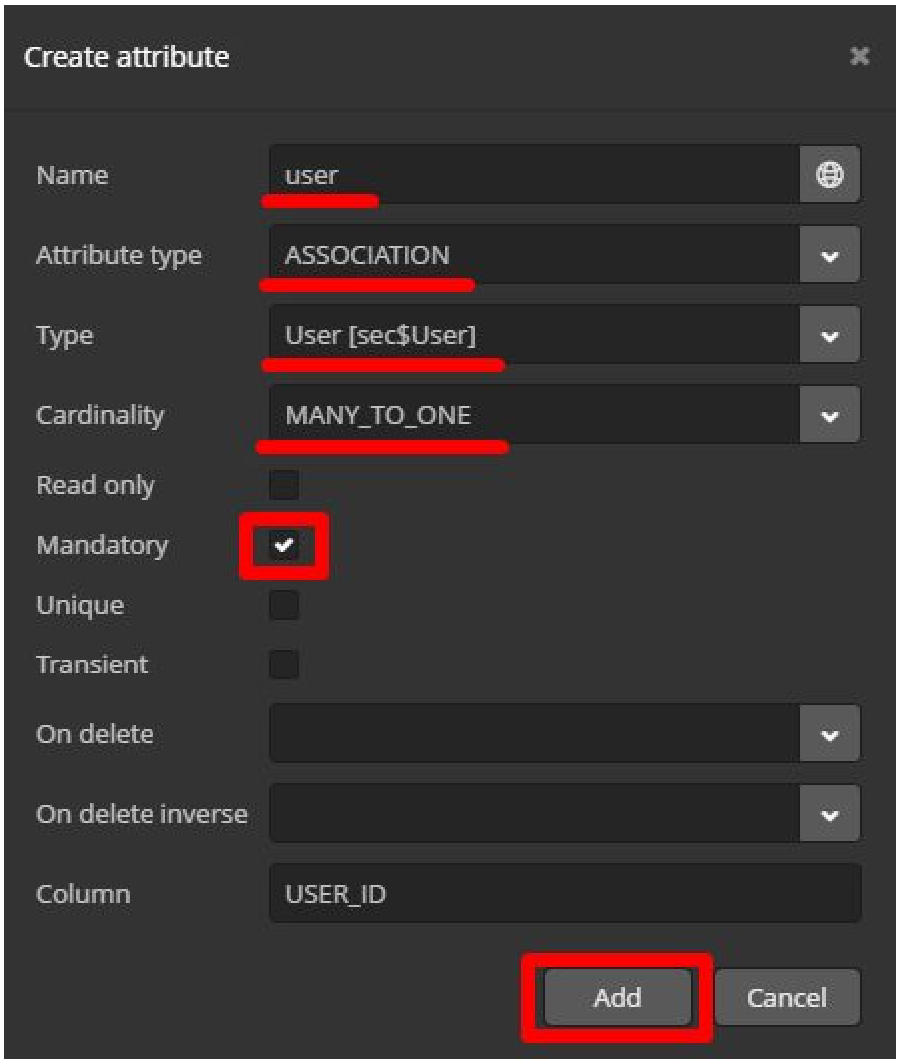
Name pattern attributes: name, phoneNumber

12. Clicking the **Source** and **DDL Preview** tabs you can find your Entity bean implementation annotated with the *javax.persistence* annotations and SQL script for creating the corresponding table
13. Click **OK** in the right-top corner of the entity designer screen to save the entity

3.2 Mechanic entity

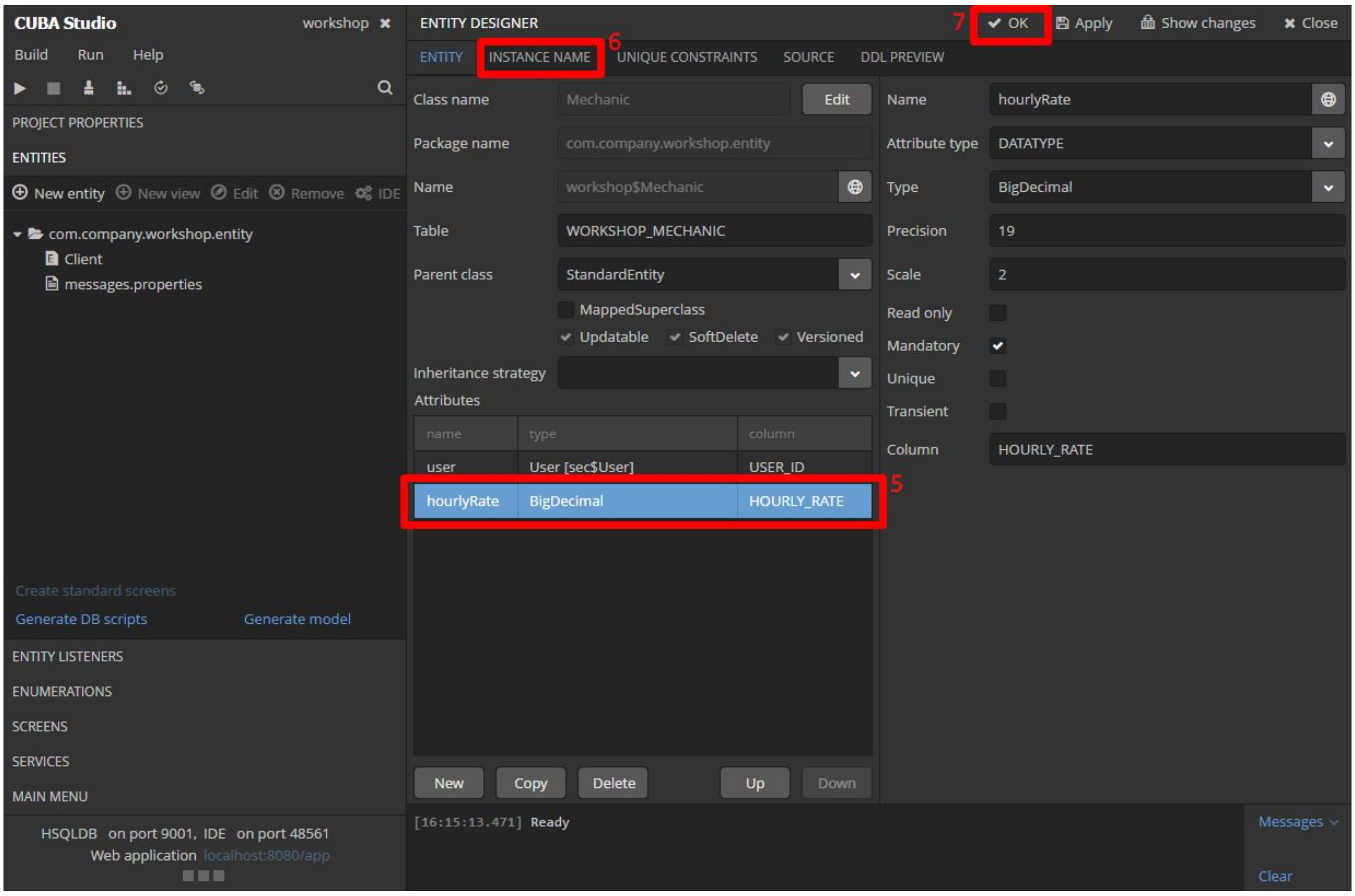
1. Create a **New entity** with **Class name**: ***Mechanic*** and click **OK**
2. Click the **New** button under the **Attributes** table
3. Fill up the **Create attribute** form as following and click **OK**
   Name: ***user***
   Attribute type: ***Association***
   Type: ***User [sec$User]***
   Cardinality: *MANY_TO_ONE*

Mandatory: Yes


Create attribute

Name	user
Attribute type	ASSOCIATION
Type	User [sec$User]
Cardinality	MANY_TO_ONE
Read only	
Mandatory	✓
Unique	
Transient	
On delete	
On delete inverse	
Column	USER_ID

4. Create one more attribute
5. Apply the following settings and click **Add**:
   Name: ***hourlyRate***
   Type: *BigDecimal*
   Mandatory: Yes
6. Move to the **Instance name** tab and select ***user*** as an instance name for Mechanic
7. Check that that your *Mechanic* entity corresponds to the the picture below and click **OK**



HSQLDB on port 9001, IDE on port 48561
Web application localhost:8080/app

Class name	Mechanic
Package name	com.company.workshop.entity
Name	workshop$Mechanic
Table	WORKSHOP_MECHANIC
Parent class	StandardEntity
	MappedSuperclass
	✓ Updatable ✓ SoftDelete ✓ Versioned
Inheritance strategy	

Attributes

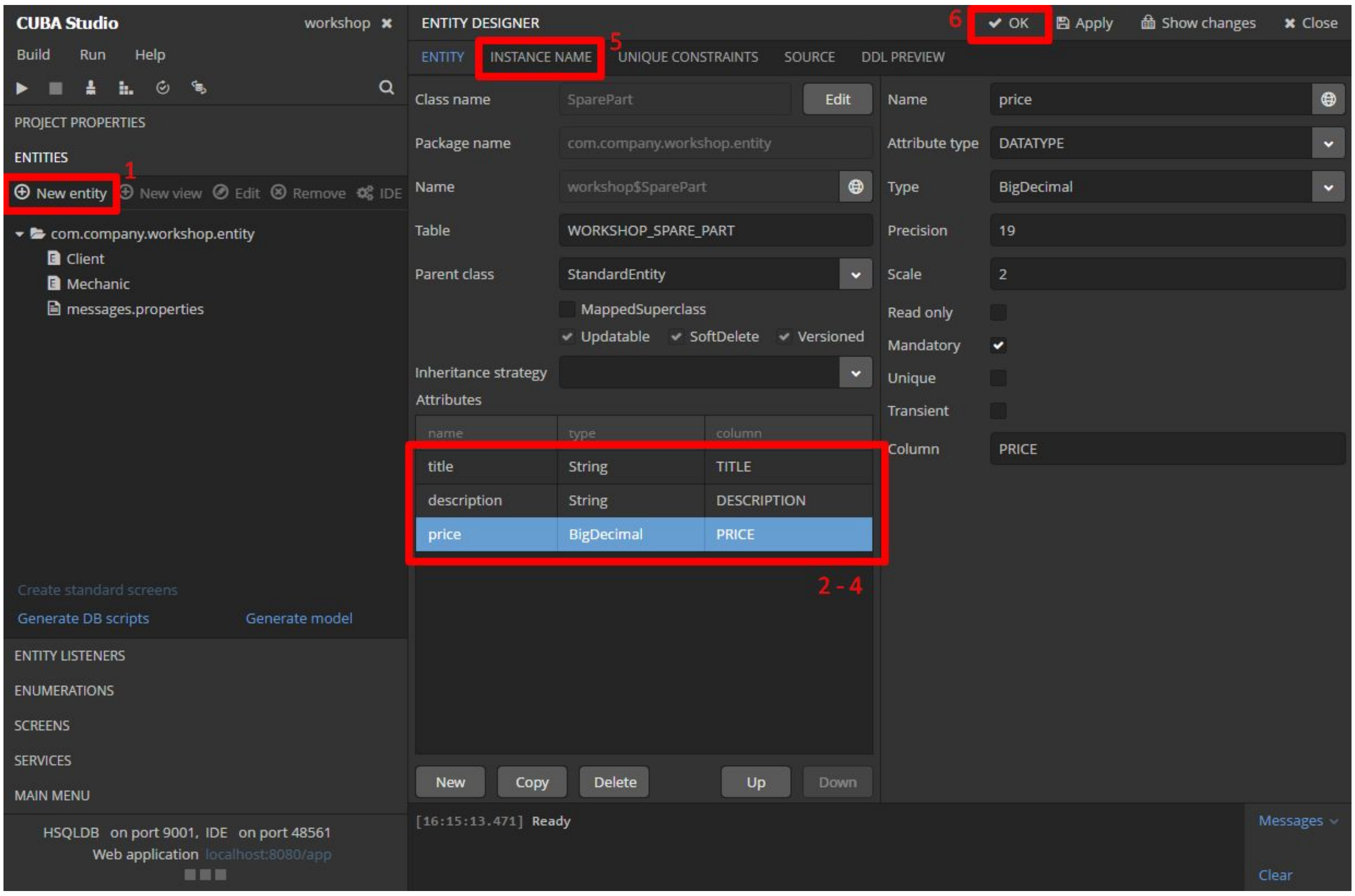
name	type	column
user	User [sec$User]	USER_ID
hourlyRate	BigDecimal	HOURLY_RATE

Name	hourlyRate
Attribute type	DATATYPE
Type	BigDecimal
Precision	19
Scale	2
Read only	
Mandatory	✓
Unique	
Transient	
Column	HOURLY_RATE

```
[16:15:13.471] Ready
```

3.3 SparePart Entity

1. Create a **New entity** with **Class name: *SparePart***
2. Add an attribute
   Name: *title*
   Type: *String*
   Mandatory: Yes
   Unique: Yes
3. Add an attribute
   Name: *description*
   Type: *String*
   Length: Unlimited
4. Add an attribute
   Name: *price*
   Type: *BigDecimal*
   Mandatory: Yes
5. Go to the **Instance name** tab and select the **title** attribute as the entity instance name
6. Check that that your *SparePart* entity corresponds to the the picture below and click **OK**



Class name	SparePart
Package name	com.company.workshop.entity
Name	workshop$SparePart
Table	WORKSHOP_SPARE_PART
Parent class	StandardEntity
MappedSuperclass	
Updatable	✓
SoftDelete	✓
Versioned	✓
Inheritance strategy	

Attributes

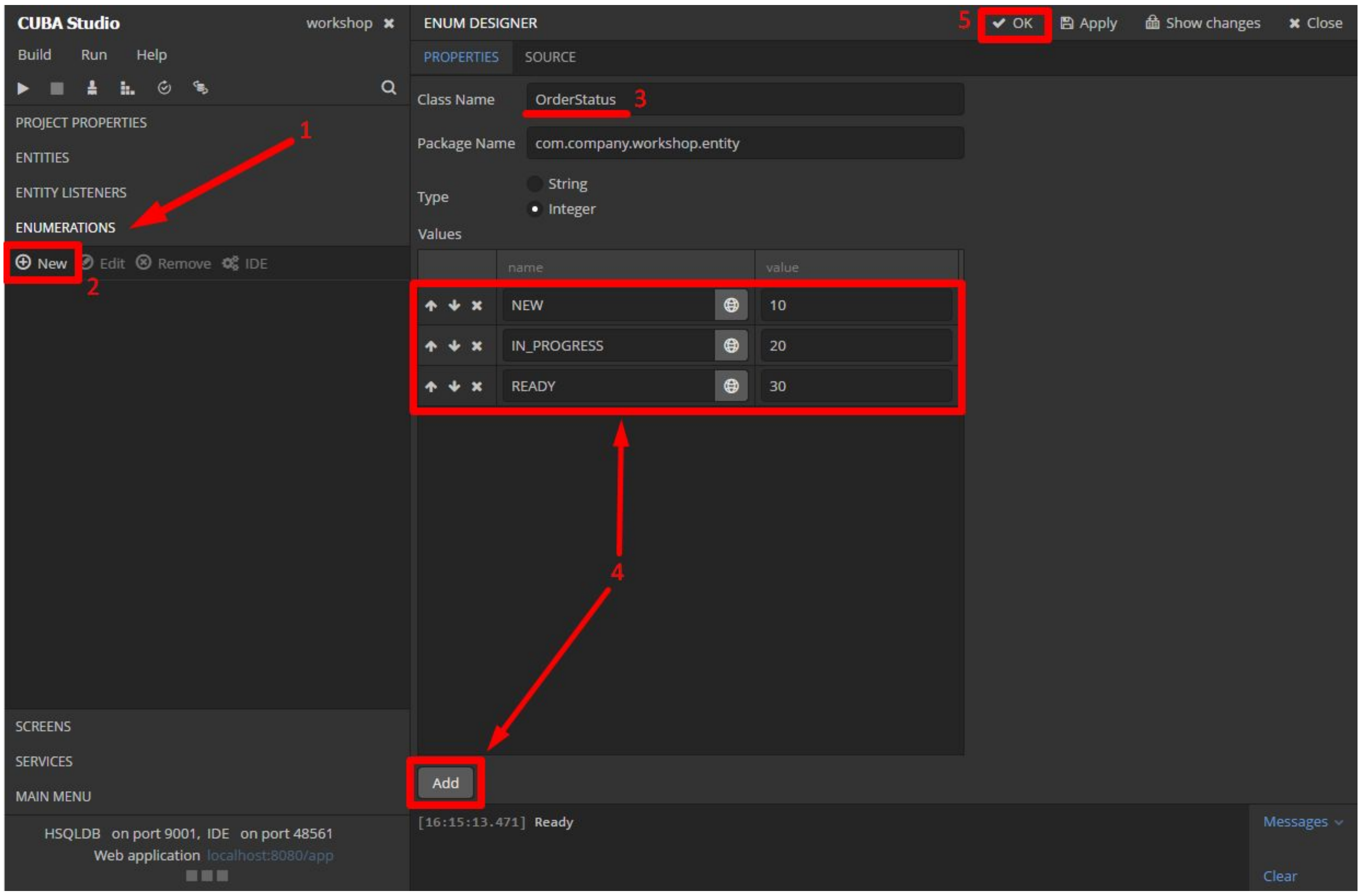
name	type	column
title	String	TITLE
description	String	DESCRIPTION
price	BigDecimal	PRICE

Name	price
Attribute type	DATATYPE
Type	BigDecimal
Precision	19
Scale	2
Read only	
Mandatory	✓
Unique	
Transient	
Column	PRICE

```
[16:15:13.471] Ready
```

3.4 OrderStatus Enum

1. Go to the **Enumerations** section in the left hand side navigation panel
2. Click **New**
3. Enter **Class name**: ***OrderStatus***
4. Add values:
   NEW 10
   IN_PROGRESS 20
   READY 30
5. Check that that your *OrderStatus* enum corresponds to the the picture below and click **OK**



Class Name: OrderStatus

Package Name: com.company.workshop.entity

Type: String [✓ Integer]

Values

name	value
NEW	10
IN_PROGRESS	20
READY	30

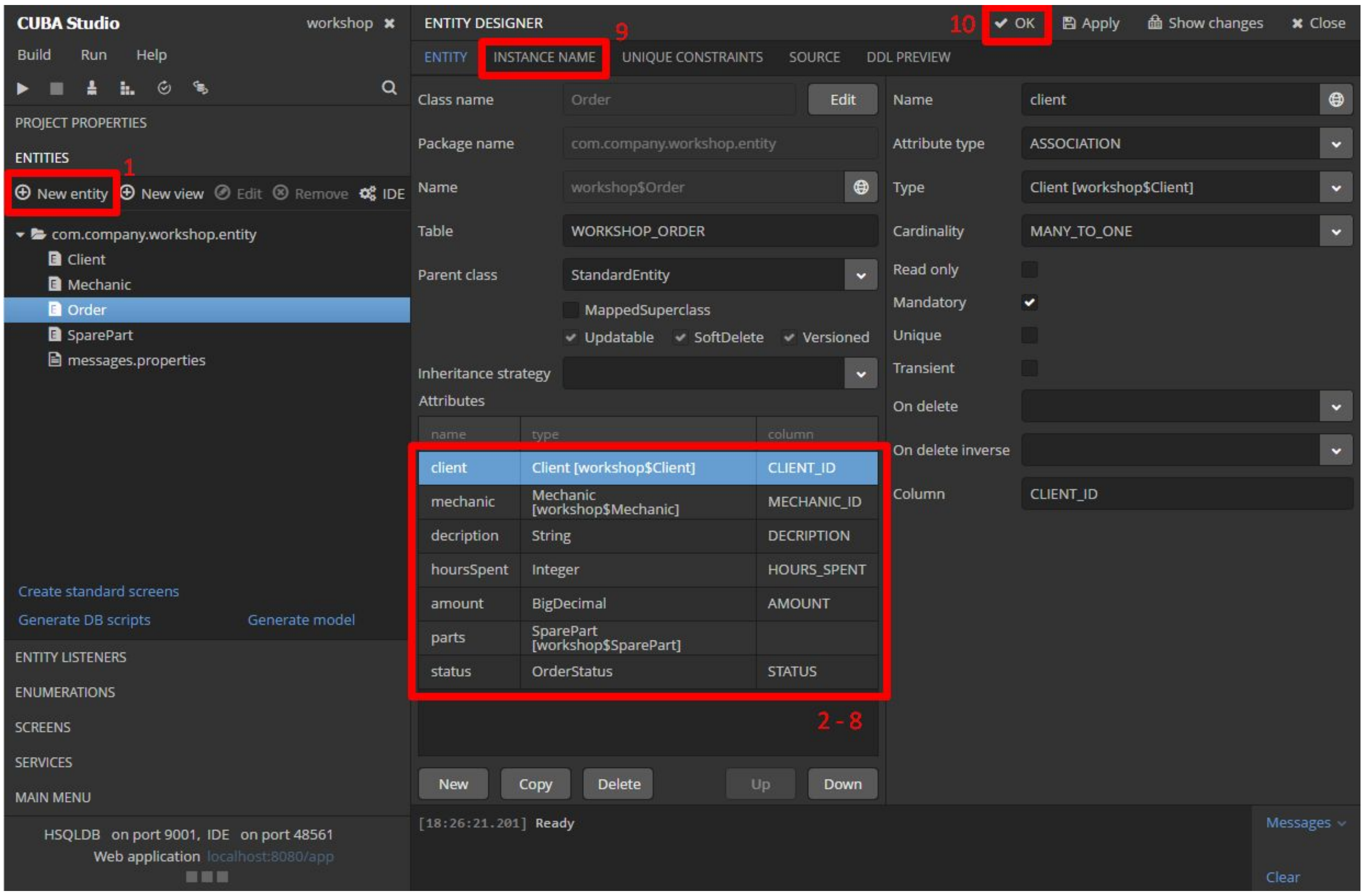
```
[16:15:13.471] Ready
```

HSQLDB on port 9001, IDE on port 48561
Web application localhost:8080/app

3.5 Order Entity

1. Create a **New entity** with **Class name:** ***Order***
2. Add an attribute
   Name: ***client***
   Attribute type: ***ASSOCIATION***
   Type: ***Client***
   Cardinality: MANY_TO_ONE
   Mandatory: Yes
3. Add an attribute
   Name: ***mechanic***
   Attribute type: ***ASSOCIATION***
   Type: ***Mechanic***
   Cardinality: MANY_TO_ONE
   Mandatory: Yes
4. Add an attribute
   Name: ***description***
   Type: ***String***
   Length: Unlimited
5. Add an attribute
   Name: ***hoursSpent***
   Type: ***Integer***

6. Add an attribute
   Name: *amount*
   Type: *BigDecimal*
7. Add an attribute
   Name: *parts*
   Attribute type: *ASSOCIATION*
   Type:SparePart
   Cardinality: MANY_TO_MANY
   The Studio will offer to create a inverse attribute - click **No**
8. Add an attribute
   Name: *status*
   Attribute type: *Enum*
   Type: *OrderStatus*
9. Set the **Instance name** for the *Order* entity to its **description** attribute
10. Check that that your *Order* entity corresponds to the the picture below and click **OK**



Class name: Order
Package name: com.company.workshop.entity
Name: workshop$Order
Table: WORKSHOP_ORDER
Parent class: StandardEntity
MappedSuperclass
✓ Updatable ✓ SoftDelete ✓ Versioned
Inheritance strategy:
Attributes

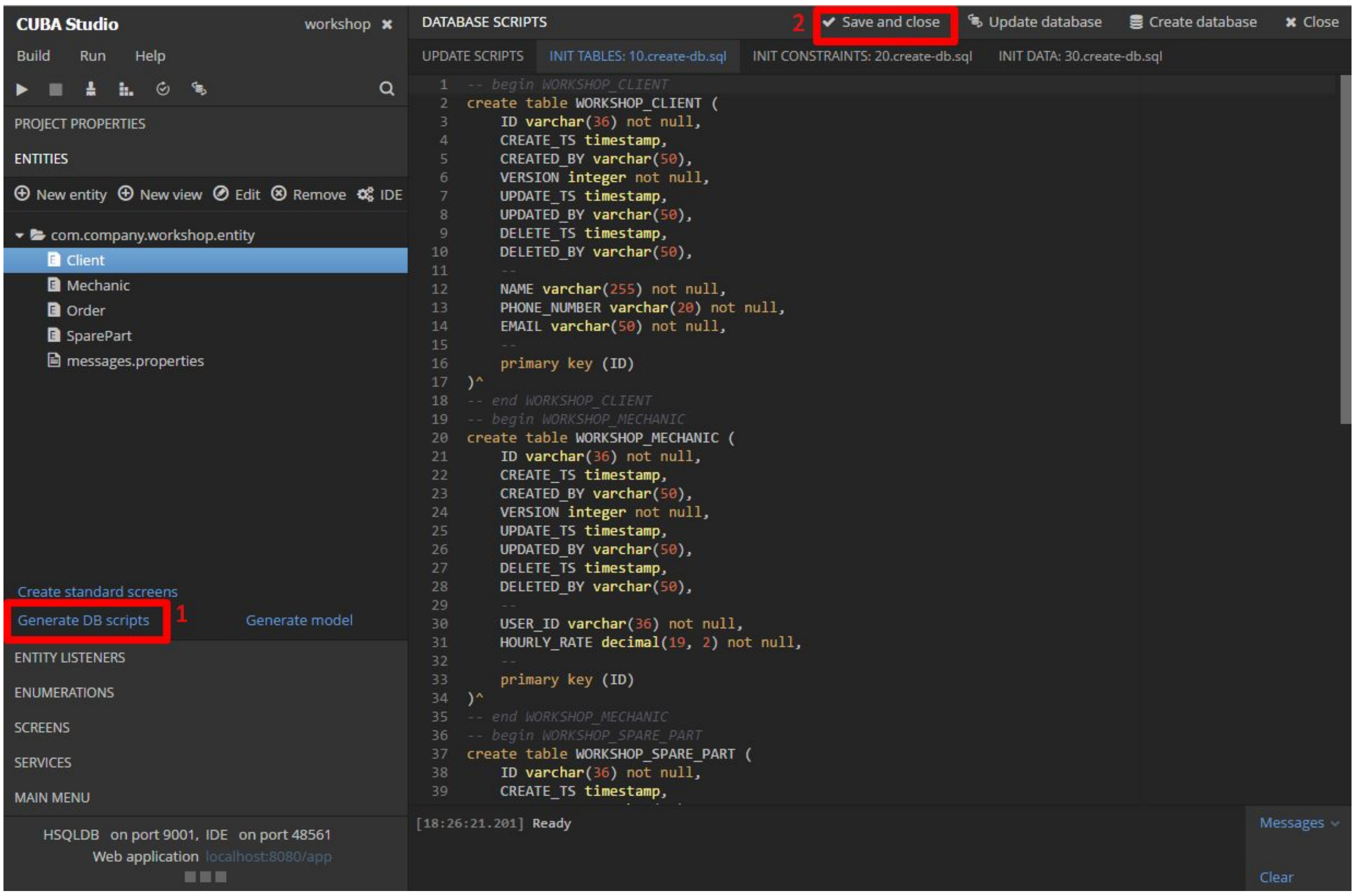
name	type	column
client	Client [workshop$Client]	CLIENT_ID
mechanic	Mechanic [workshop$Mechanic]	MECHANIC_ID
decription	String	DECRIPTION
hoursSpent	Integer	HOURS_SPENT
amount	BigDecimal	AMOUNT
parts	SparePart [workshop$SparePart]	
status	OrderStatus	STATUS

Name: client
Attribute type: ASSOCIATION
Type: Client [workshop$Client]
Cardinality: MANY_TO_ONE
Read only:
Mandatory: ✓
Unique:
Transient:
On delete:
On delete inverse:
Column: CLIENT_ID

```
[18:26:21.201] Ready
```

3.6 Generate DB Scripts and Create the Database

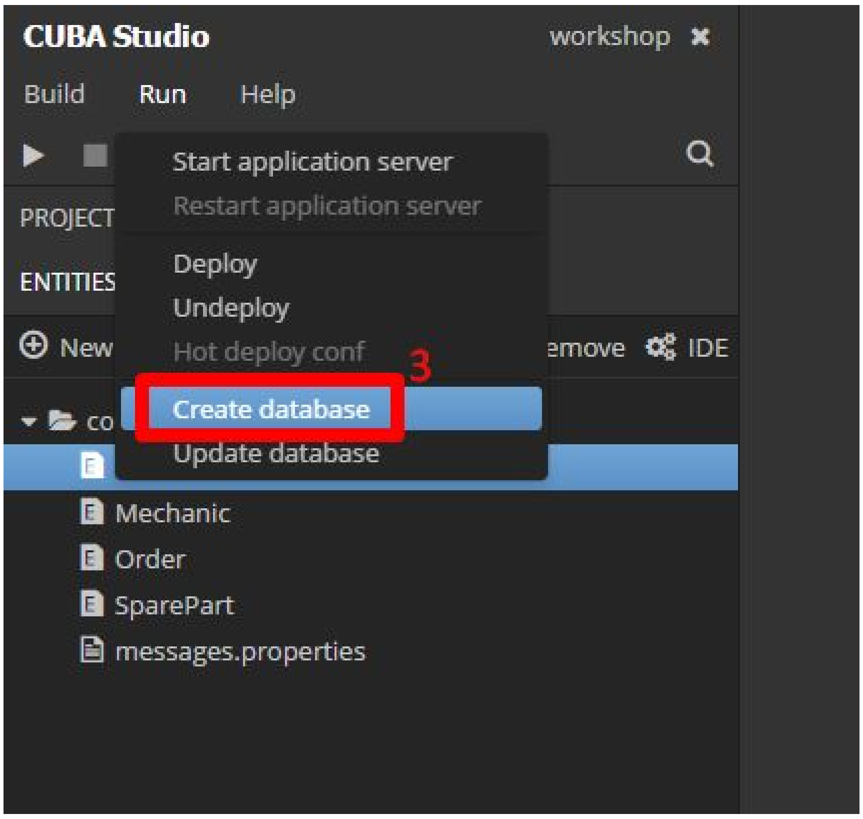
1. Click the **Generate DB scripts** link In the bottom of the **Entities** section; the CUBA Studio will generate a script to create tables and constraints
2. Click **Save and close** and the Studio will save the scripts into a special directory of our project, so we will be able to access them if needed



```sql
-- begin WORKSHOP_CLIENT
create table WORKSHOP_CLIENT (
    ID varchar(36) not null,
    CREATE_TS timestamp,
    CREATED_BY varchar(50),
    VERSION integer not null,
    UPDATE_TS timestamp,
    UPDATED_BY varchar(50),
    DELETE_TS timestamp,
    DELETED_BY varchar(50),
    --
    NAME varchar(255) not null,
    PHONE_NUMBER varchar(20) not null,
    EMAIL varchar(50) not null,
    --
    primary key (ID)
)^
-- end WORKSHOP_CLIENT
-- begin WORKSHOP_MECHANIC
create table WORKSHOP_MECHANIC (
    ID varchar(36) not null,
    CREATE_TS timestamp,
    CREATED_BY varchar(50),
    VERSION integer not null,
    UPDATE_TS timestamp,
    UPDATED_BY varchar(50),
    DELETE_TS timestamp,
    DELETED_BY varchar(50),
    --
    USER_ID varchar(36) not null,
    HOURLY_RATE decimal(19, 2) not null,
    --
    primary key (ID)
)^
-- end WORKSHOP_MECHANIC
-- begin WORKSHOP_SPARE_PART
create table WORKSHOP_SPARE_PART (
    ID varchar(36) not null,
    CREATE_TS timestamp,
```

```
[18:26:21.201] Ready
```

3. Invoke the **Run — Create database** action from the menu to create a database



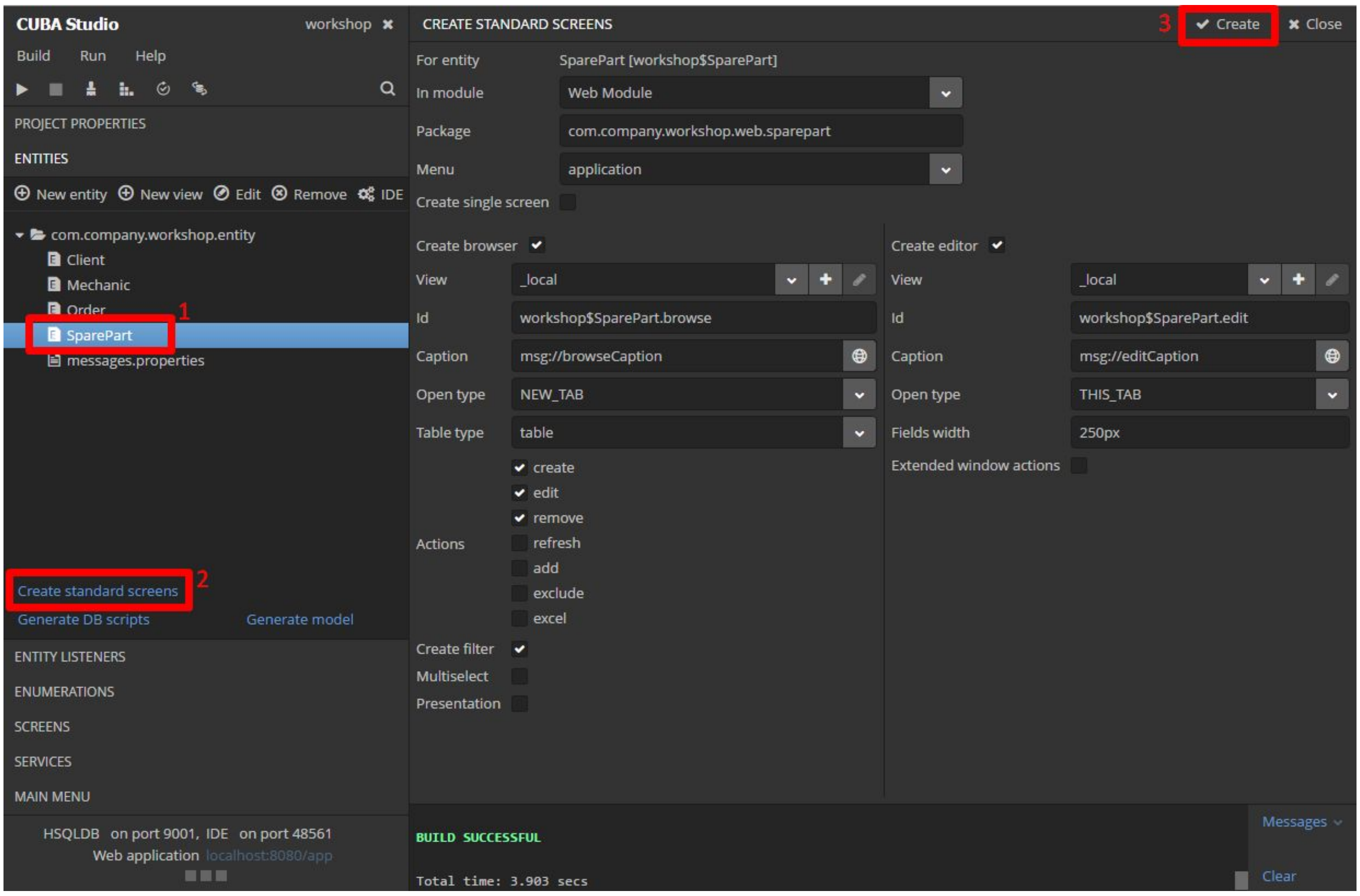
4. The CUBA Studio warns us that the old DB will be deleted, click **OK**

4 AUTO-GENERATING CRUD UI

Under CRUD or standard UI we understand screens that allow you to browse the list of records (browser) and create/edit one record (editor).

4.1 SparePart CRUD UI

1. Open the **Entities** section of the left hand side navigation panel and select the **SparePart** entity
2. Click on the **Create standard screens** link.
   Create standard screens dialog appears. It allows you to specify where to place the screens in the project, what menu item to be used to access the browser screen, should it be a single combined screen for all operations or two separate screens to read the list and create, update and delete an instance
3. Keep the default values and click **Create**

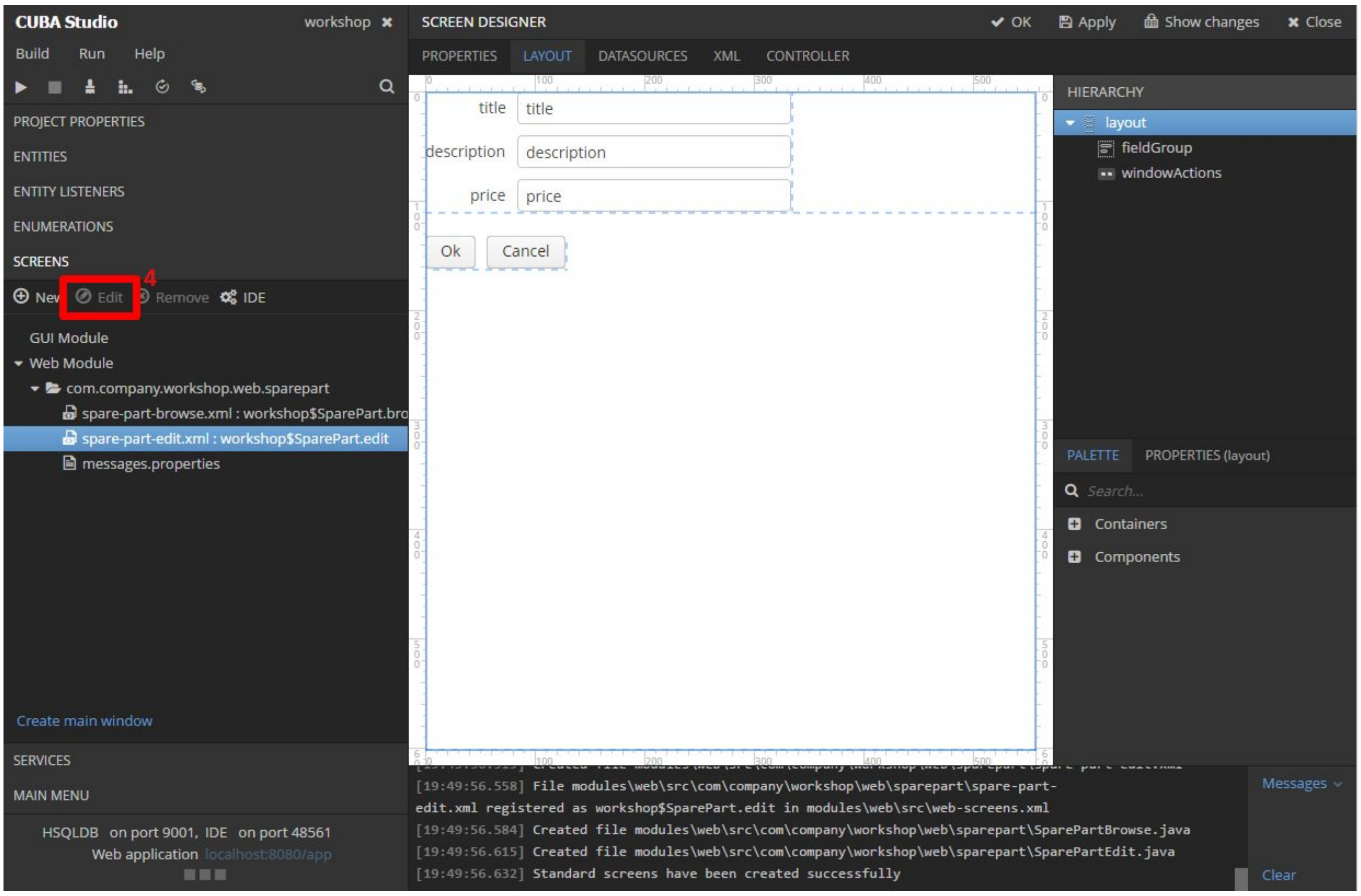


For entity: SparePart [workshop$SparePart]
In module: Web Module
Package: com.company.workshop.web.sparepart
Menu: application
Create single screen: ☐

Create browser	✓	Create editor	✓
View	_local	View	_local
Id	workshop$SparePart.browse	Id	workshop$SparePart.edit
Caption	msg://browseCaption	Caption	msg://editCaption
Open type	NEW_TAB	Open type	THIS_TAB
Table type	table	Fields width	250px
Actions	✓ create, ✓ edit, ✓ remove, ☐ refresh, ☐ add, ☐ exclude, ☐ excel	Extended window actions	☐
Create filter	✓		
Multiselect	☐		
Presentation	☐		

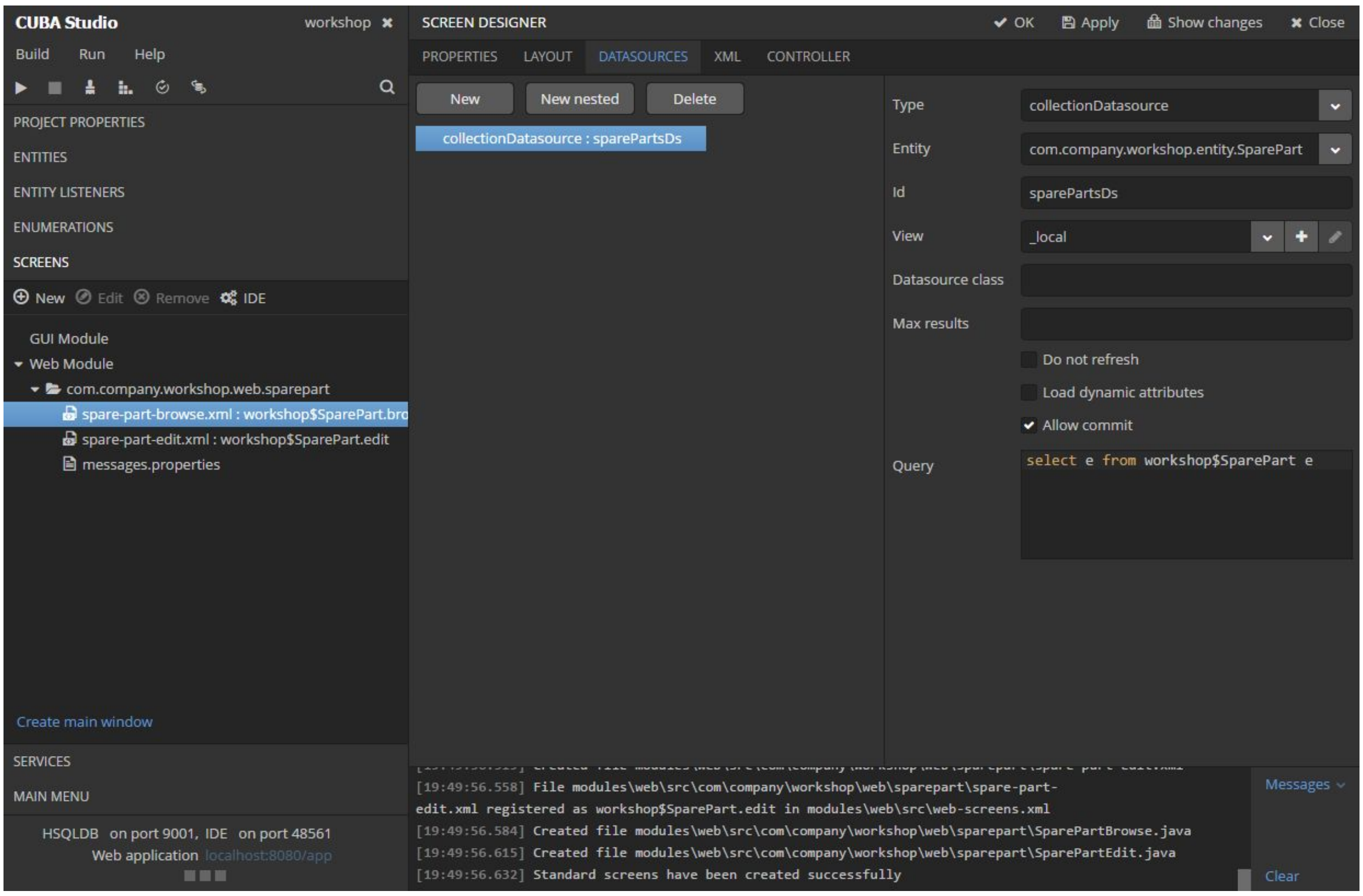
```
BUILD SUCCESSFUL

Total time: 3.903 secs
```

4. The studio has generated separate screens: browser and editor. You can pick one of them and click **Edit** in the **Screens** section of the navigation panel. Screen designer has WYSIWYG editor, XML and Controller tab to be able to see screen declaration and its Java controller.



```
[19:49:56.558] File modules\web\src\com\company\workshop\web\sparepart\spare-part-
edit.xml registered as workshop$SparePart.edit in modules\web\src\web-screens.xml
[19:49:56.584] Created file modules\web\src\com\company\workshop\web\sparepart\SparePartBrowse.java
[19:49:56.615] Created file modules\web\src\com\company\workshop\web\sparepart\SparePartEdit.java
[19:49:56.632] Standard screens have been created successfully
```

5. Visual components that tightly work with data (tables, dropdowns, etc) are data-aware and connect to the database through datasources. Taking SparePart browser as an example, let's have a look at data binding. Select the **sparepart-browse.xml** screen, click **Edit** and open the **Datasources** tab.
   Datasources use standard JPQL queries to load data. In our example the result list of this query will be automatically shown in the main table of the screen, as ***sparePartsDs*** is set to the datasoure property of the table.

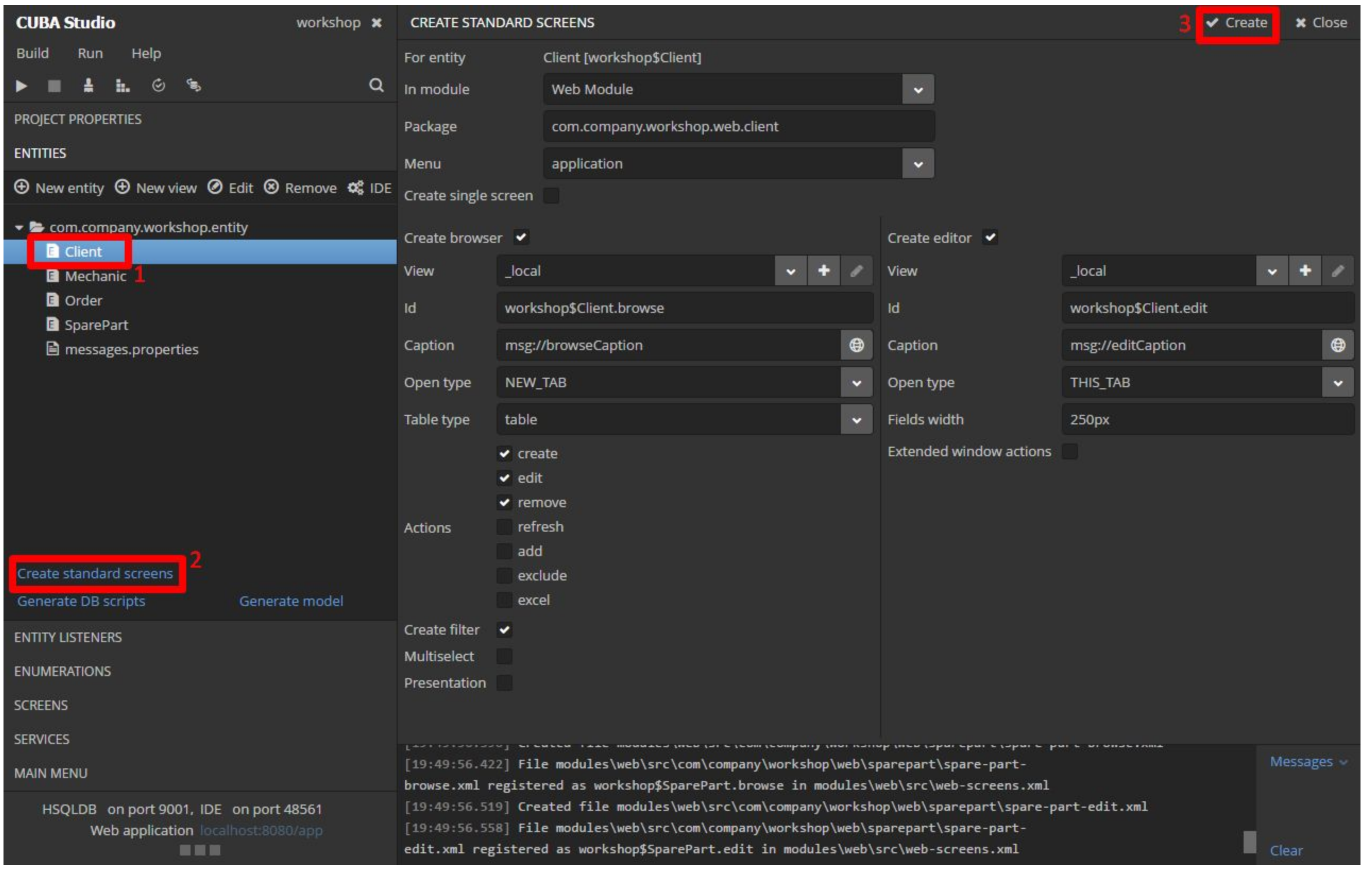


Type: collectionDatasource
Entity: com.company.workshop.entity.SparePart
Id: sparePartsDs
View: _local
Datasource class:
Max results:
Do not refresh
Load dynamic attributes
[✓ Allow commit]
Query: select e from workshop$SparePart e

```
[19:49:56.558] File modules\web\src\com\company\workshop\web\sparepart\spare-part-
edit.xml registered as workshop$SparePart.edit in modules\web\src\web-screens.xml
[19:49:56.584] Created file modules\web\src\com\company\workshop\web\sparepart\SparePartBrowse.java
[19:49:56.615] Created file modules\web\src\com\company\workshop\web\sparepart\SparePartEdit.java
[19:49:56.632] Standard screens have been created successfully
```

4.2 Client CRUD UI

1. Open the **Entities** section of the navigation panel and select the **Client** entity
2. Click **Create standard screens**
3. Click **Create**



For entity	Client [workshop$Client]
In module	Web Module
Package	com.company.workshop.web.client
Menu	application
Create single screen	☐

Create browser	✓	Create editor	✓
View	_local	View	_local
Id	workshop$Client.browse	Id	workshop$Client.edit
Caption	msg://browseCaption	Caption	msg://editCaption
Open type	NEW_TAB	Open type	THIS_TAB
Table type	table	Fields width	250px
		Extended window actions	☐

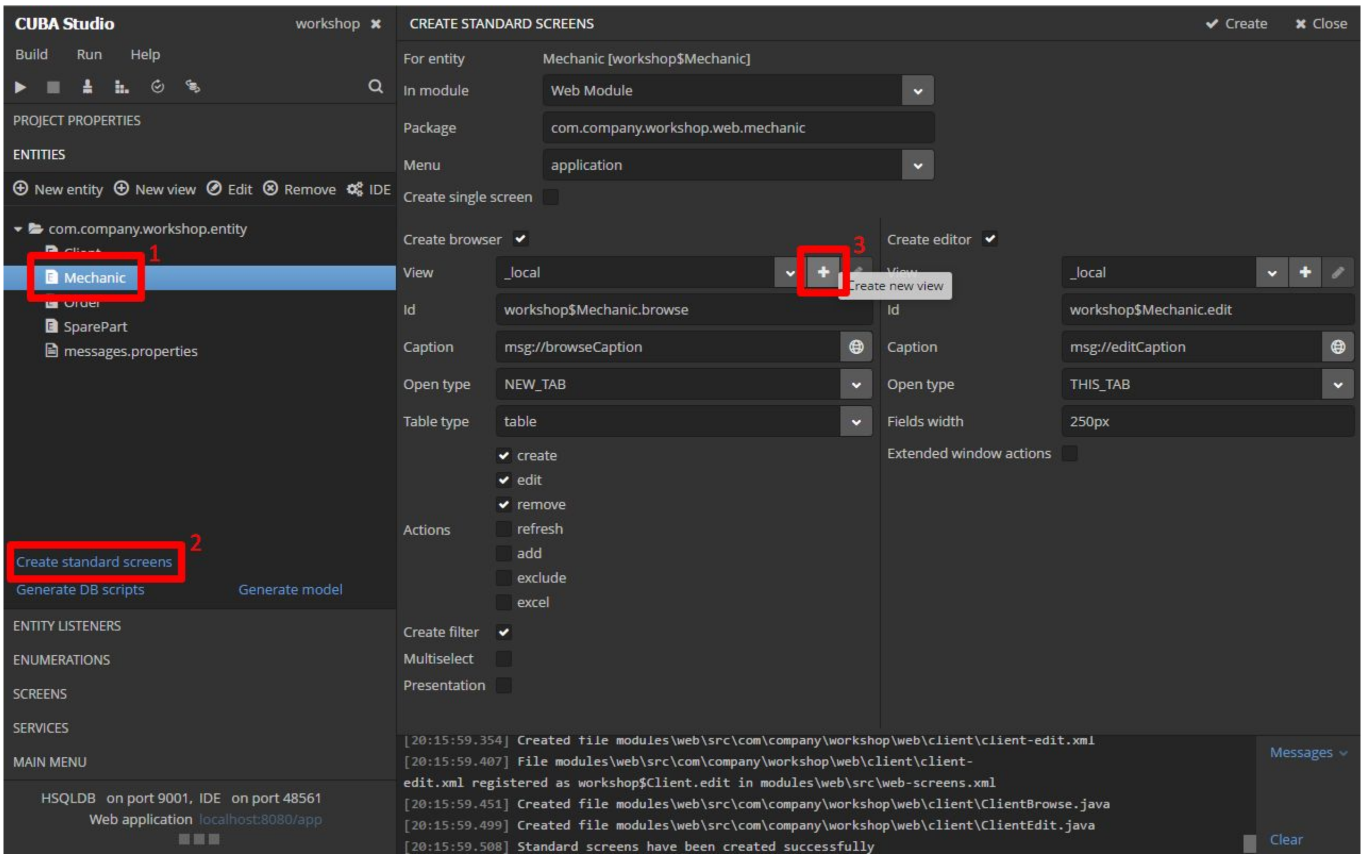
Actions: ✓ create, ✓ edit, ✓ remove, ☐ refresh, ☐ add, ☐ exclude, ☐ excel

Create filter ✓
Multiselect ☐
Presentation ☐

```
[19:49:56.422] File modules\web\src\com\company\workshop\web\sparepart\spare-part-
browse.xml registered as workshop$SparePart.browse in modules\web\src\web-screens.xml
[19:49:56.519] Created file modules\web\src\com\company\workshop\web\sparepart\spare-part-edit.xml
[19:49:56.558] File modules\web\src\com\company\workshop\web\sparepart\spare-part-
edit.xml registered as workshop$SparePart.edit in modules\web\src\web-screens.xml
```

4.3 Mechanic CRUD UI

1. Open the **Entities** section of the navigation panel and select the **Mechanic** entity
2. Click **Create standard screens**
3. Mechanic differs from the entities we have just created UI for. It has relation to the system **User** entity, which is used in CUBA for security reasons. So, our datasource should load the **user** field along with the simple fields of the mechanic entity. For that purpose CUBA uses views - an XML description of what entity attributes should be loaded from the database.
   Click the **plus** button to create a new view and the **View designer screen** will appear

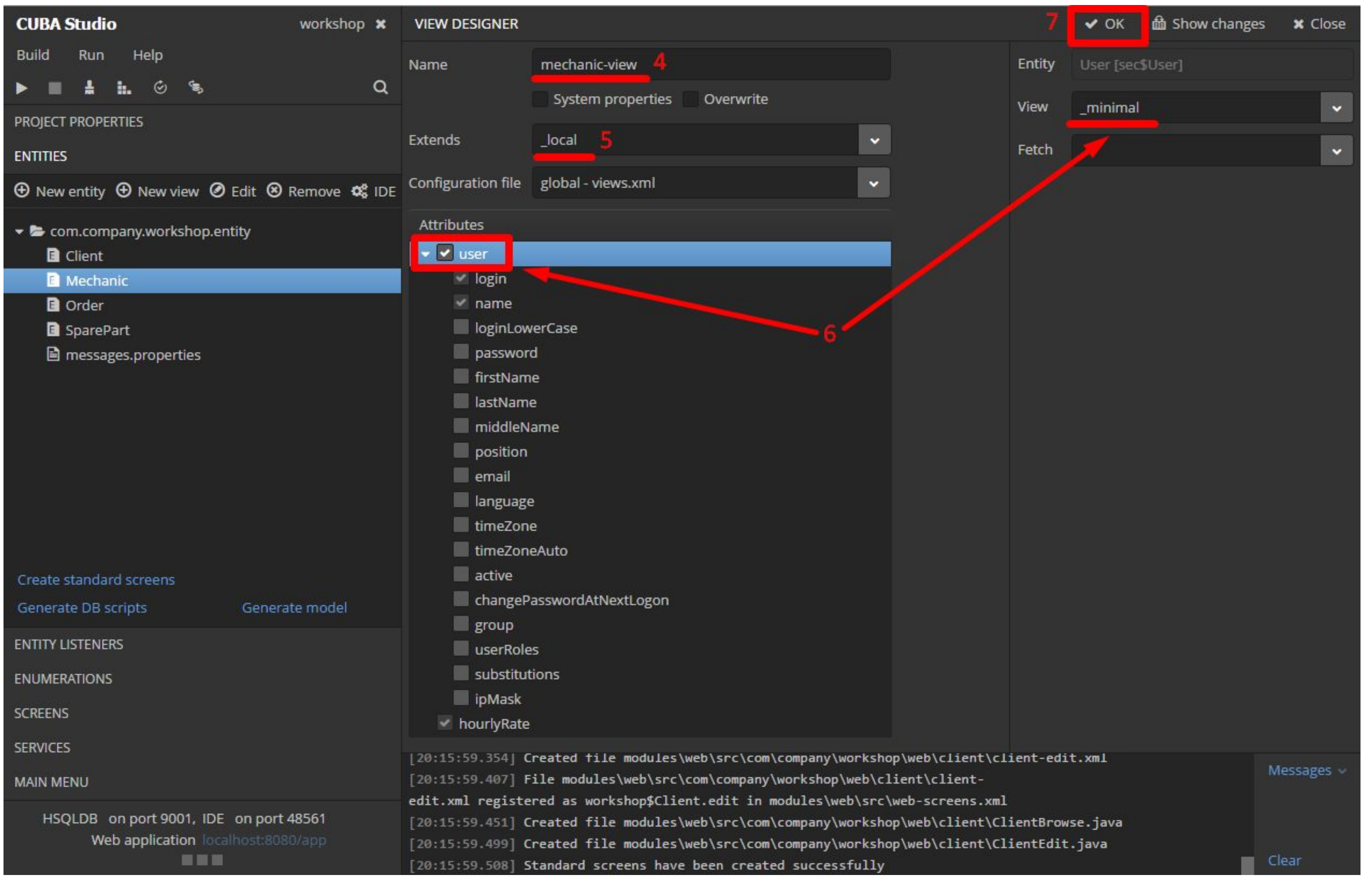


For entity	Mechanic [workshop$Mechanic]
In module	Web Module
Package	com.company.workshop.web.mechanic
Menu	application
Create single screen	☐

Create browser	✓	Create editor	✓
View	_local	View	_local
Id	workshop$Mechanic.browse	Id	workshop$Mechanic.edit
Caption	msg://browseCaption	Caption	msg://editCaption
Open type	NEW_TAB	Open type	THIS_TAB
Table type	table	Fields width	250px
Actions	✓ create, ✓ edit, ✓ remove, ☐ refresh, ☐ add, ☐ exclude, ☐ excel	Extended window actions	☐
Create filter	✓		
Multiselect	☐		
Presentation	☐		

```
[20:15:59.354] Created file modules\web\src\com\company\workshop\web\client\client-edit.xml
[20:15:59.407] File modules\web\src\com\company\workshop\web\client\client-
edit.xml registered as workshop$Client.edit in modules\web\src\web-screens.xml
[20:15:59.451] Created file modules\web\src\com\company\workshop\web\client\ClientBrowse.java
[20:15:59.499] Created file modules\web\src\com\company\workshop\web\client\ClientEdit.java
[20:15:59.508] Standard screens have been created successfully
```

4. Specify the **Name**: ***mechanic-view***, as we will use it for both browser and editor
5. Our view extends the ***_local*** one, which is the system view that includes all the simple (local) attributes.
   By default all screens use *_local* view for standard screen generation. It means that references to other entities will not be loaded and shown by default.
6. Select **user** attribute and specify the ***_minimal*** system view for it; _minimal view includes only attributes, that are specified in the **Instance Name** of an entity. This view will give us enough information to show an entity is the browser's table and editor's field group
7. Click **OK** to save the view



Name: mechanic-view

System properties ☐ Overwrite ☐

Extends: _local

Configuration file: global - views.xml

Entity: User [sec$User]

View: _minimal

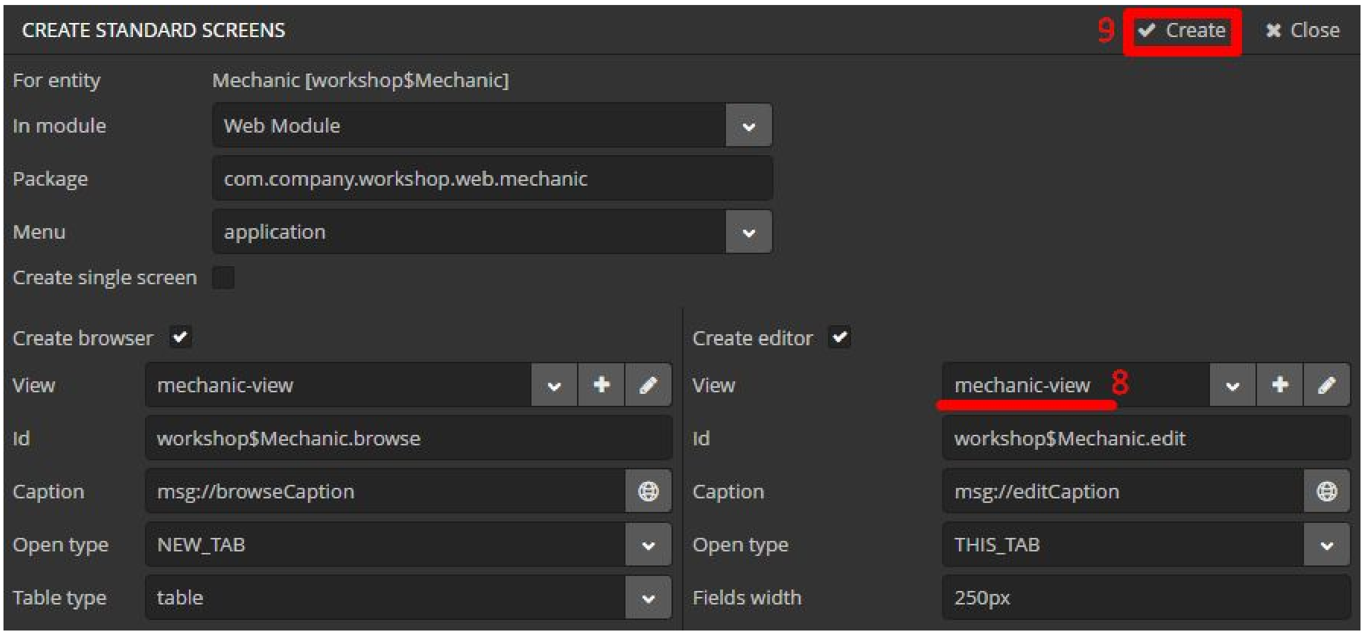
Fetch:

Attributes

- ✓ user
  - ✓ login
  - ✓ name
  - ☐ loginLowerCase
  - ☐ password
  - ☐ firstName
  - ☐ lastName
  - ☐ middleName
  - ☐ position
  - ☐ email
  - ☐ language
  - ☐ timeZone
  - ☐ timeZoneAuto
  - ☐ active
  - ☐ changePasswordAtNextLogon
  - ☐ group
  - ☐ userRoles
  - ☐ substitutions
  - ☐ ipMask
- ✓ hourlyRate

```
[20:15:59.354] Created file modules\web\src\com\company\workshop\web\client\client-edit.xml
[20:15:59.407] File modules\web\src\com\company\workshop\web\client\client-
edit.xml registered as workshop$Client.edit in modules\web\src\web-screens.xml
[20:15:59.451] Created file modules\web\src\com\company\workshop\web\client\ClientBrowse.java
[20:15:59.499] Created file modules\web\src\com\company\workshop\web\client\ClientEdit.java
[20:15:59.508] Standard screens have been created successfully
```

8. Specify the same view for the editor
9. Click **Create**



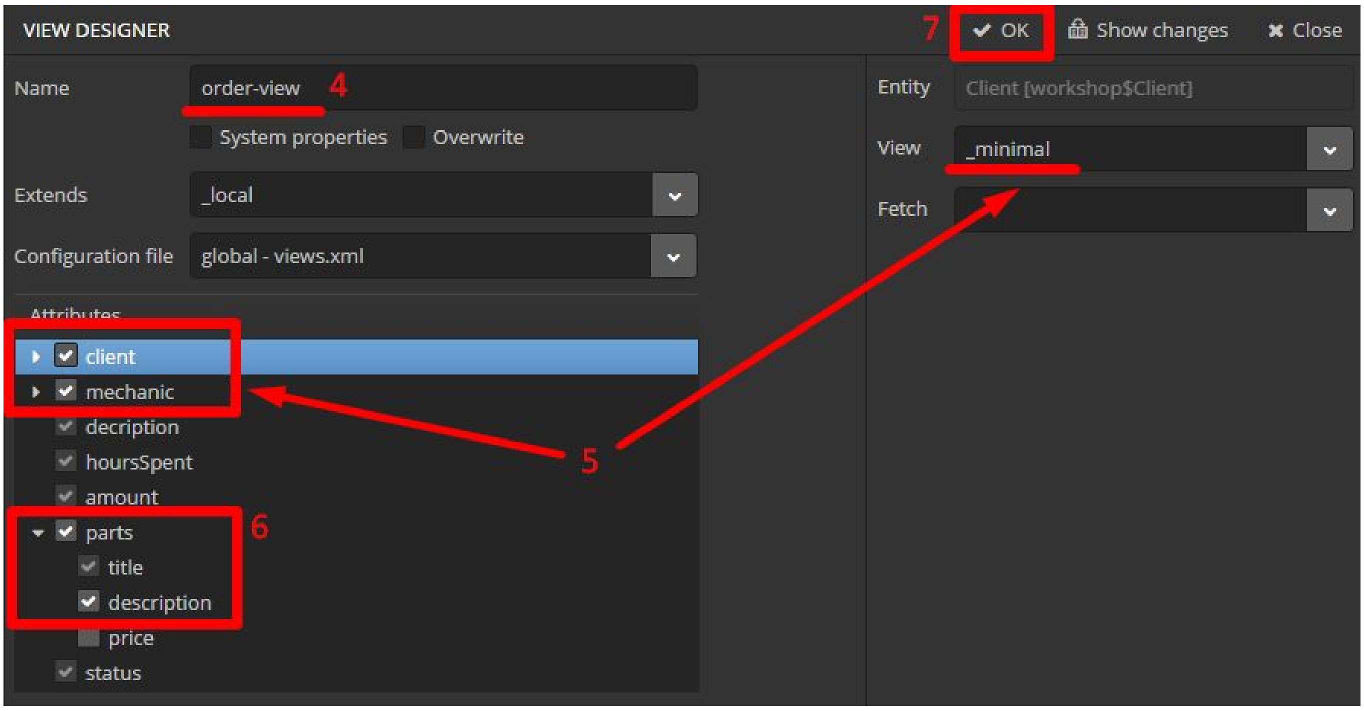
For entity	Mechanic [workshop$Mechanic]
In module	Web Module
Package	com.company.workshop.web.mechanic
Menu	application
Create single screen	☐

Create browser ✓		Create editor ✓	
View	mechanic-view	View	mechanic-view
Id	workshop$Mechanic.browse	Id	workshop$Mechanic.edit
Caption	msg://browseCaption	Caption	msg://editCaption
Open type	NEW_TAB	Open type	THIS_TAB
Table type	table	Fields width	250px

4.4 Order CRUD UI

1. Open the **Entities** section of the navigation panel and select the **Order** entity
2. Click **Create standard screens**

3. Similarly to what we did for the Mechanic entity, click the plus button to create an extended view for the **Order** entity
4. Rename the view to ***order-view***
5. Select **client** and **mechanic** attributes and specify the ***_minimal*** view for both of them
6. Select the **parts** attribute, which contains a collection of the SpareParts instances. Adding parts to an order we would like to see only **title** and **description** attributes, so tick them
7. Click **OK**



Name: order-view

System properties Overwrite

Extends: _local

Configuration file: global - views.xml

Attributes

- ✓ client
- ✓ mechanic
- ✓ decription
- ✓ hoursSpent
- ✓ amount
- ✓ parts
  - ✓ title
  - ✓ description
  - price
- ✓ status

Entity: Client [workshop$Client]

View: _minimal

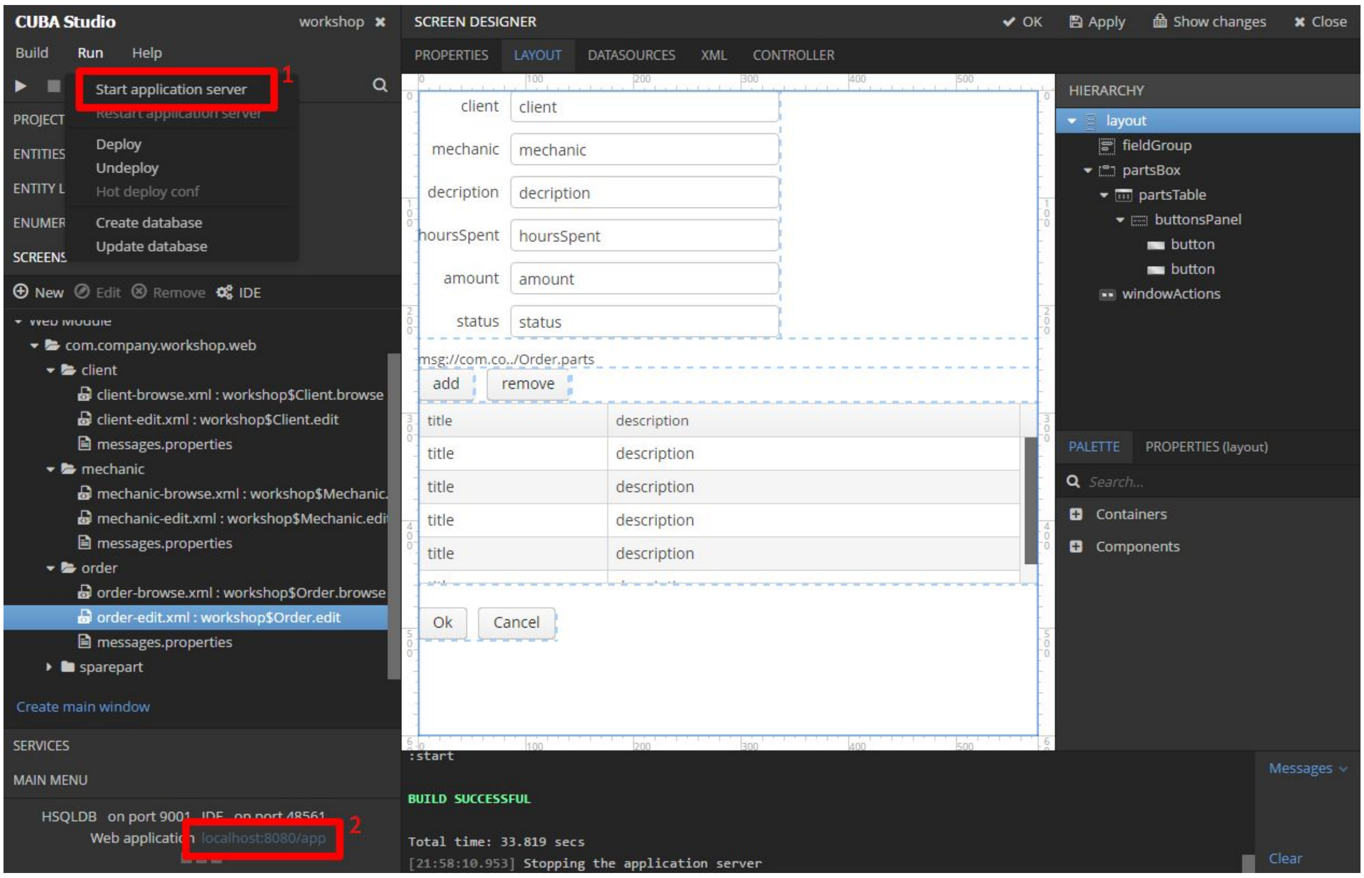
Fetch:

8. Select ***order-view*** also for the editor screen
9. Click **Create**

5 FIRST LAUNCH

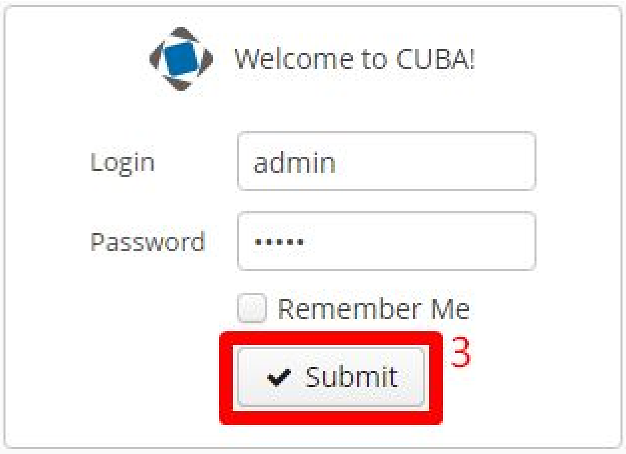
Our application is done, of course, to a first approximation. Let’s compile and launch it!

1. Invoke the **Run — Start application** action from the menu. Studio will deploy a local Tomcat instance in the project subdirectory, deploy the compiled application there and launch it
2. Open the application by clicking a link in the bottom part of the Studio

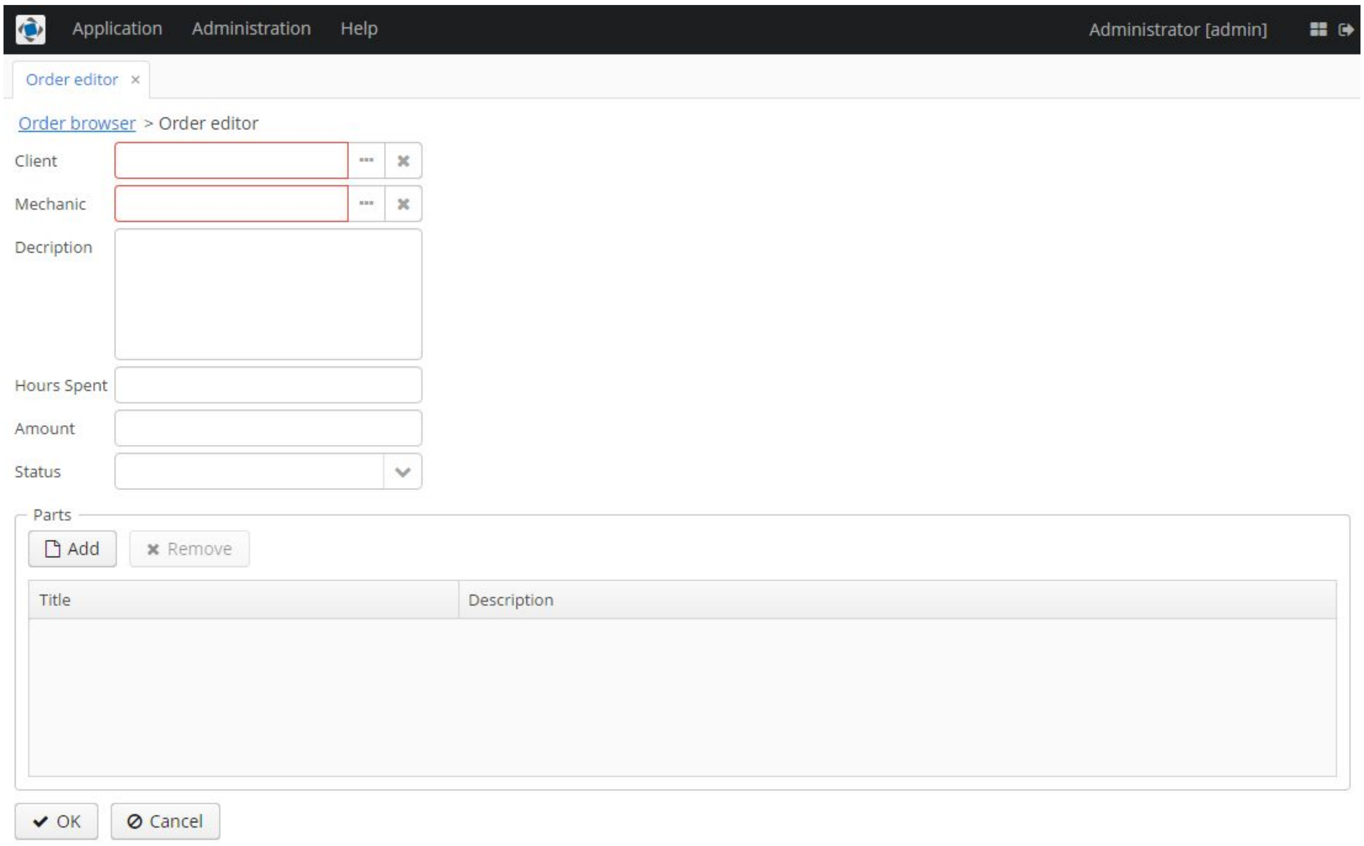


```
:start

BUILD SUCCESSFUL

Total time: 33.819 secs
[21:58:10.953] Stopping the application server
```

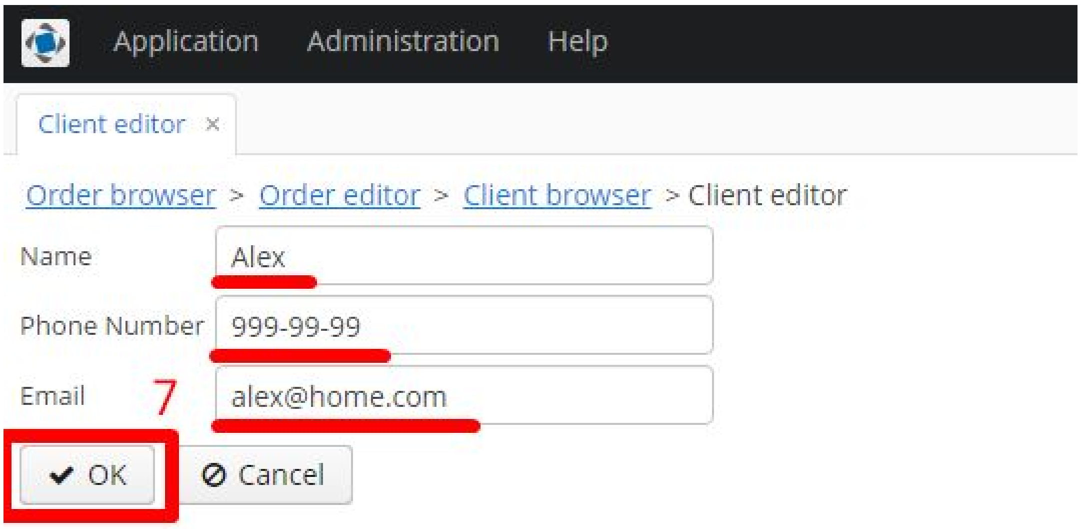
3. CUBA application welcomes us with the login screen. It's a part of the security module, integrated in the platform. Default login and password are already set in the screen, so just click **Submit**.



4. After successful login you can see the main window of your application, which can be customized from the Studio, similarly to any other screen.
5. Open **Application - Orders** from the menu. The standard browser screen appears. It contains a generic data filter on top, a table of records with buttons on top, performing standard CRUD actions. There are more standard actions available in the platform, for example export to excel. Click **Create**.
6. This is the autogenerated editor screen for the Order entity. Let's fill up the form. Click **[...]** to select a client.



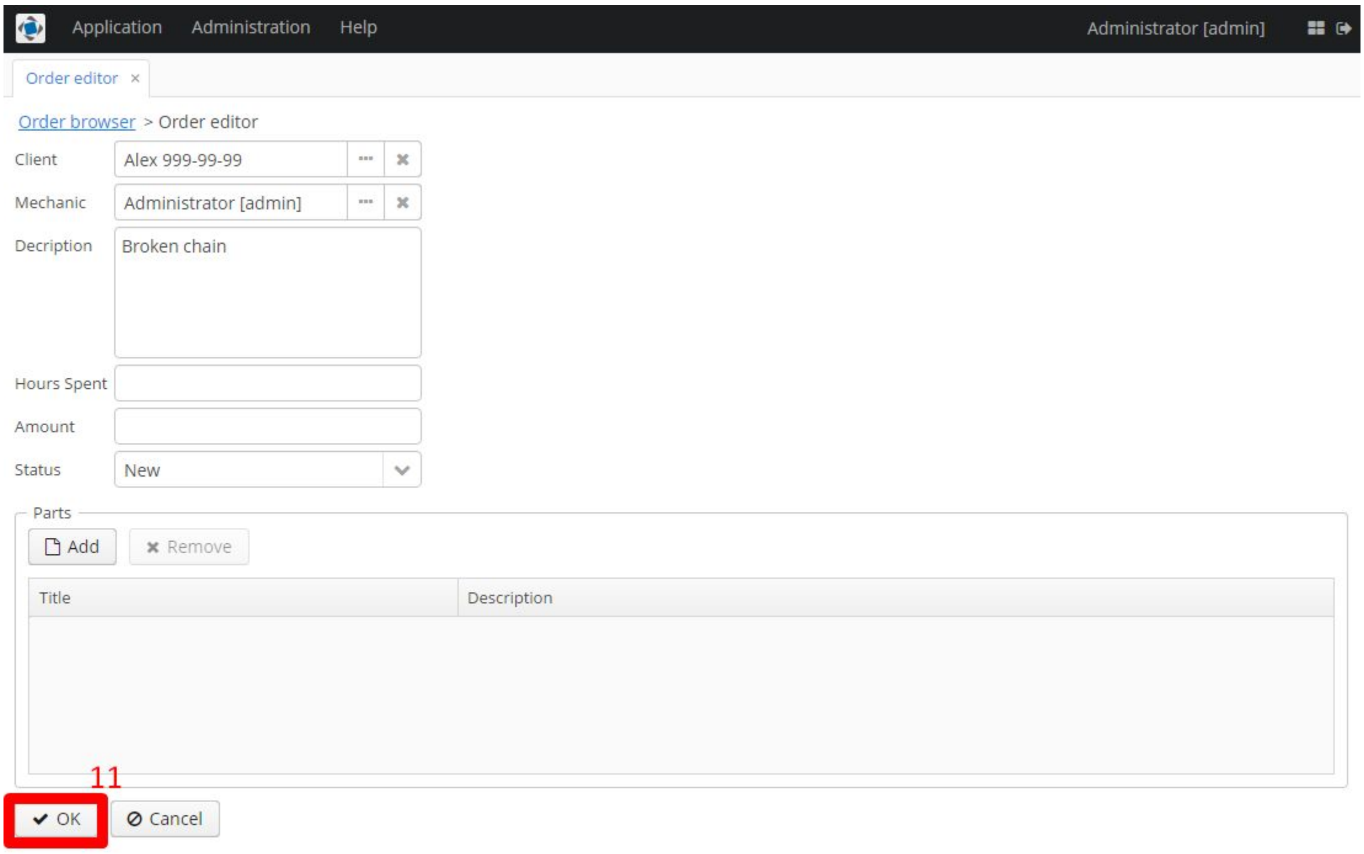
7. There are no clients in the system yet, so the client browser is empty. Click **Create** to add a new client to the system and fill up client editor with the following values and click **OK**:
   Name: ***Alex***
   Phone Number: 999-99-99
   Email: *alex@home.com*


Application
Administration
Help
Client editor ×
Order browser > Order editor > Client browser > Client editor
7
OK
Cancel

Name: Alex
Phone Number: 999-99-99
Email: alex@home.com

8. Double click the client record we have just created or simply click **Select** on the bottom part of the client browser
9. We have specified a client for our order and it is shown in the client field accordingly to the **Instance name**, we specified for the Client entity (Name Phone number)

10. Create set the Mechanic field in the same way
    User: ***admin***
    Hourly Rate: ***10***
11. Complete entering data for the order using the following values and click **OK**
    Description: ***Broken chain***
    Status: ***New***



Client: Alex 999-99-99
Mechanic: Administrator [admin]
Decription: Broken chain
Hours Spent:
Amount:
Status: New
Parts

Our CRUD application is ready, so we have successfully completed items 1-3 of the functional specification!

5.1 Generic Filter

The CUBA generic filter enables you to filter data by all the fields of the screen main entity, all the entities it refers to as well as their fields. It also gives you ability to create custom filtering rules based on JPQL queries.

Create one more order with the following parameters:

Client
Name: ***Freeman***
Phone Number: ***111-11-11***
Email: ***freeman@world.com***

Mechanic
User: ***admin***
Hourly Rate: ***8***

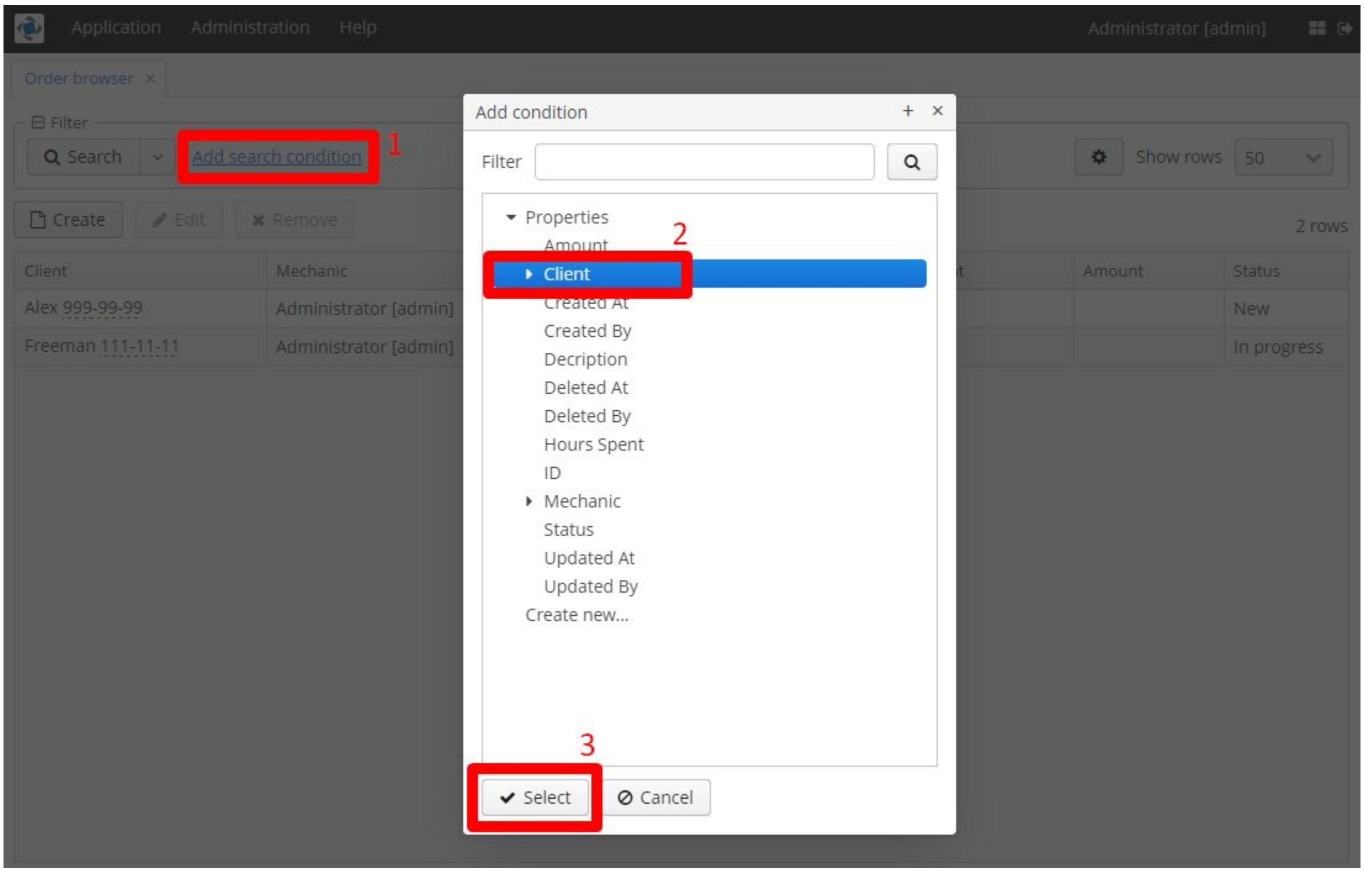
Order
Description: ***Wheels problem***
Hours Spent: ***2***
Status: ***In progress***

Parts **(add one part)**
Title: Wheel 29”
Description: Standard wheel
Price: 50

5.1.1 Client’s Orders Filter

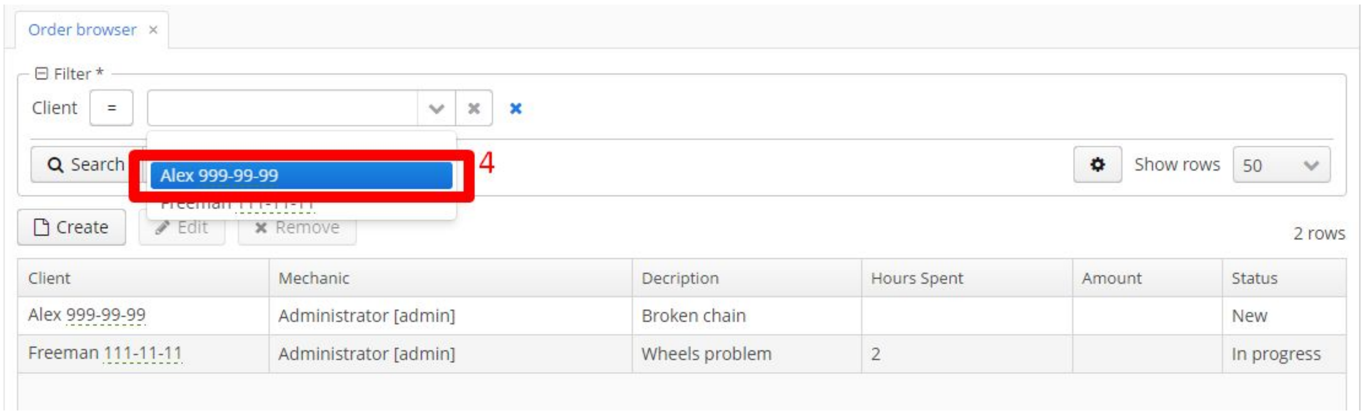
Let’s create a simple filter to find all the orders of a particular client.

1. Go to the Order Browser screen (**Application — Orders**) and click the **Add search condition** link
2. Select the **Client** property
3. Click **Select**



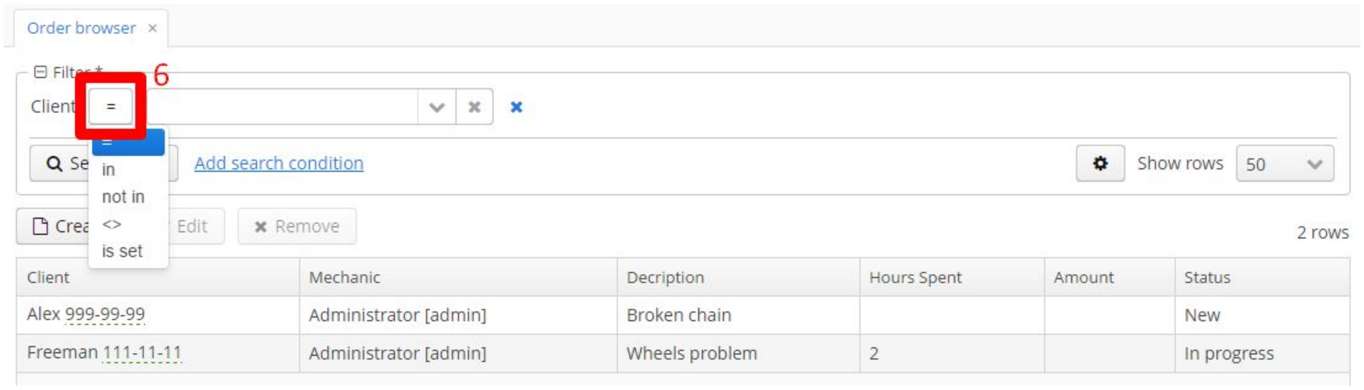
Client	Mechanic	Amount	Status
Alex 999-99-99	Administrator [admin]		New
Freeman 111-11-11	Administrator [admin]		In progress

4. Your filter automatically shows you options that can be applied for the **Client** field. Select ***Alex*** and click **Search**.



Client	Mechanic	Decription	Hours Spent	Amount	Status
Alex 999-99-99	Administrator [admin]	Broken chain			New
Freeman 111-11-11	Administrator [admin]	Wheels problem	2		In progress

5. Only one record left after filtering.
6. Click on the **[=]** button to see what comparison options CUBA filters offer for users
   [=] - exact equality
   [in] - the field is contained in a specified set of values
   [not in] - the field is NOT contained in a specified set of values
   [<>] - the field is not equal to the specified value
   [is set] - the field is null/not null



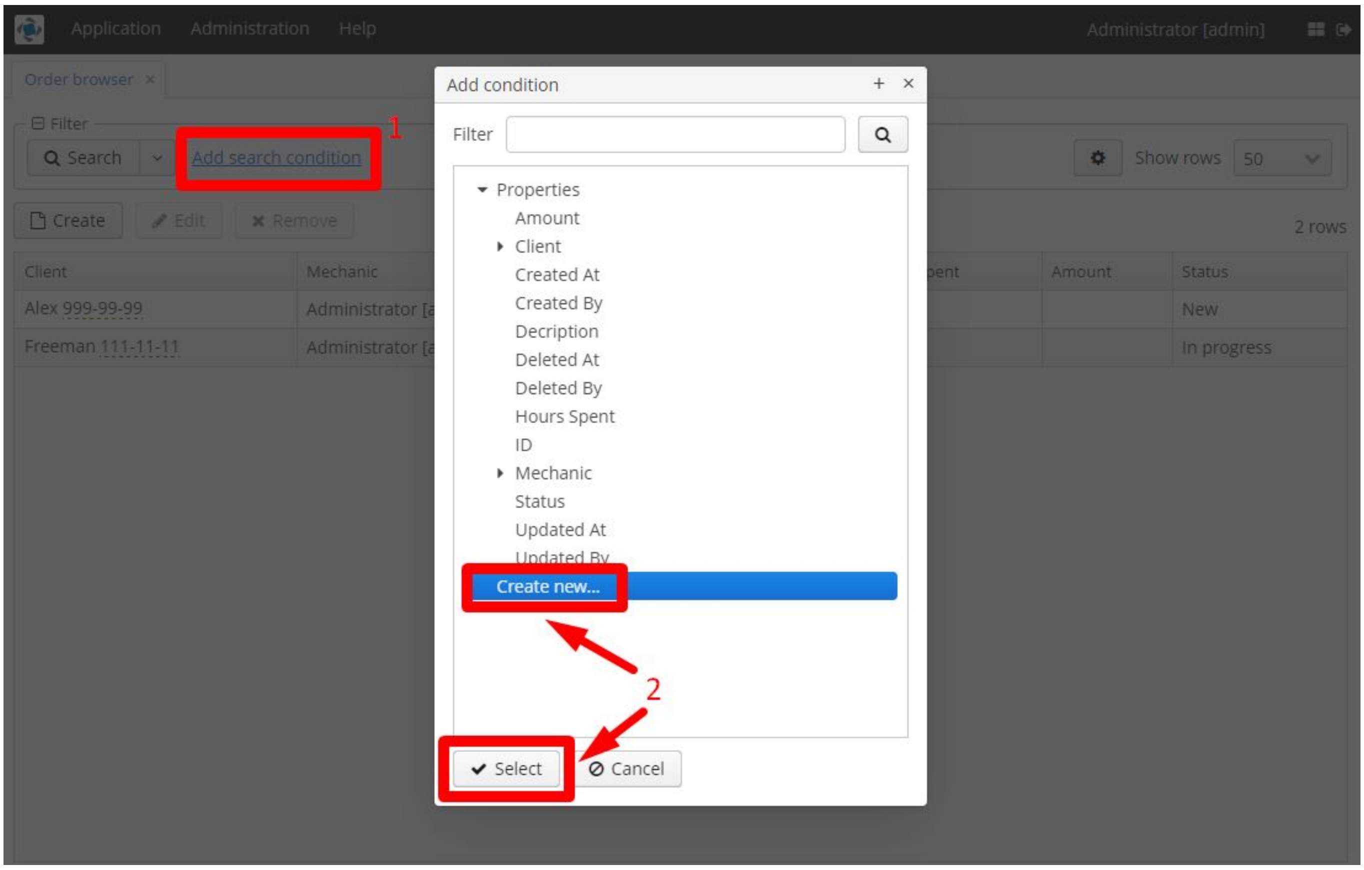
Client	Mechanic	Decription	Hours Spent	Amount	Status
Alex 999-99-99	Administrator [admin]	Broken chain			New
Freeman 111-11-11	Administrator [admin]	Wheels problem	2		In progress

In a similar way you can filter by all types of attributes (string, numeric, boolean, etc).

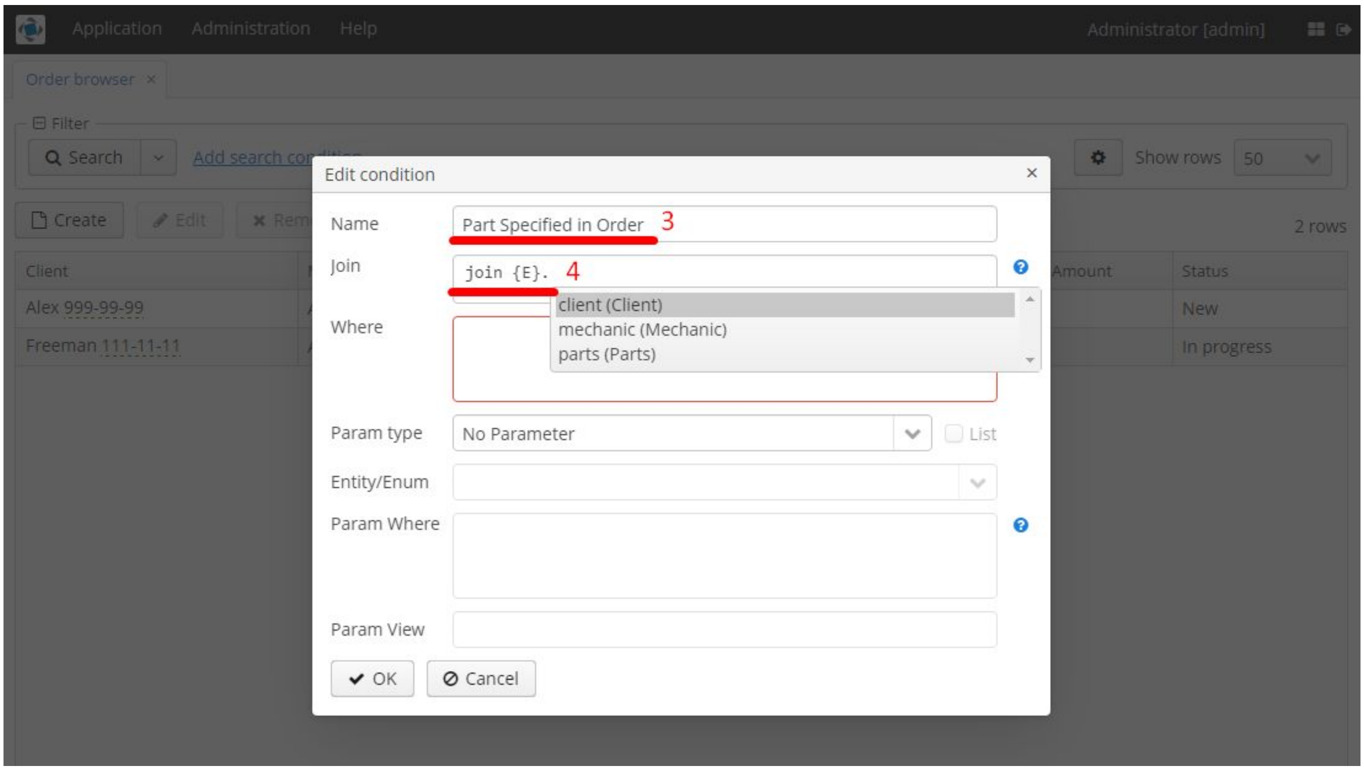
5.1.2 Part Specified in Order Filter

Sometimes it is not possible to filter records without join operation. Let’s take the following example: we want to filter all the orders that contain some particular part in it.

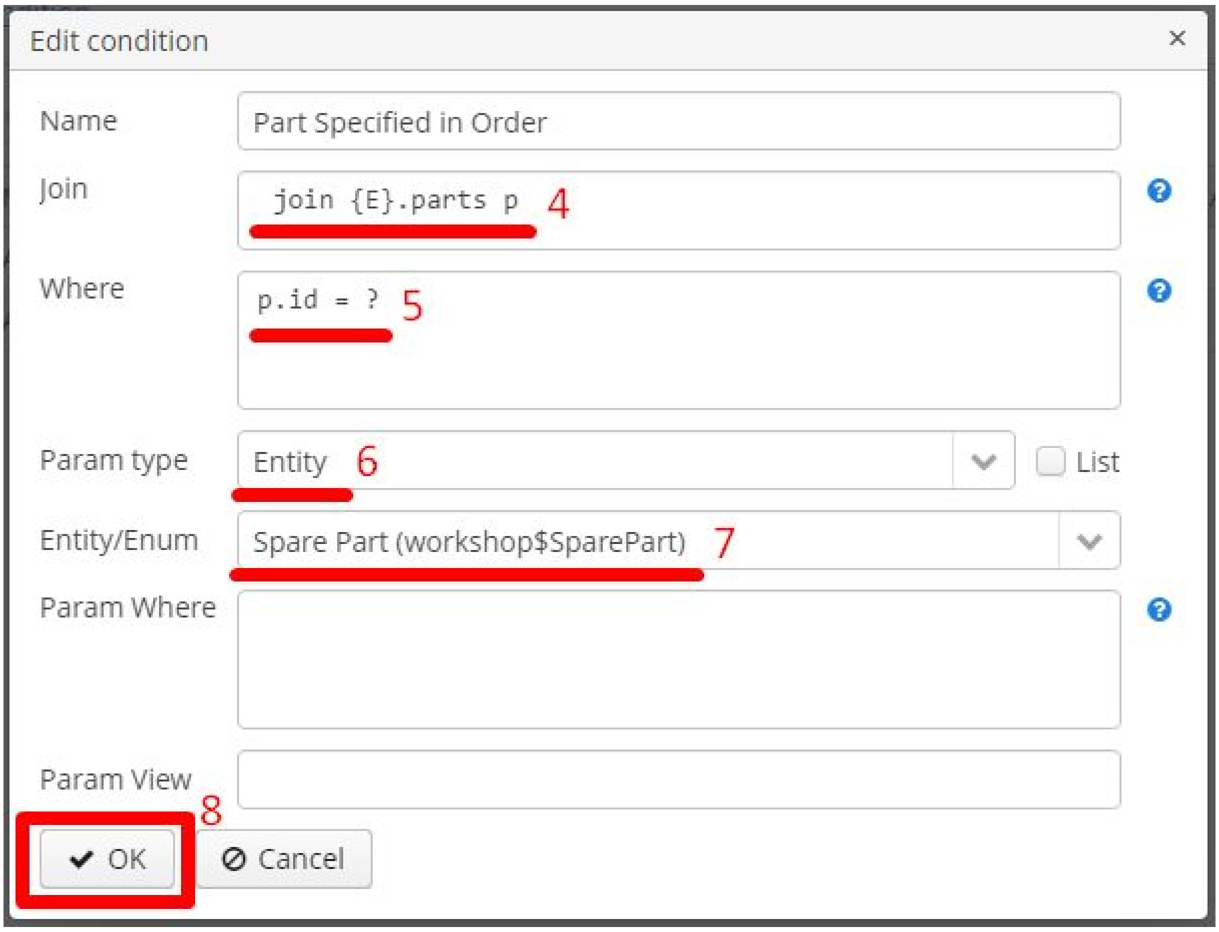
1. Click the **Add search condition** link
2. Select the **Create new…** option



3. Set condition **Name**: ***Part Specified in Order***
4. Our main entity in the screen is *Order*, which has *Set<SparePart> parts* field, containing all the used parts. Let's join parts to the orders: type ***join {E}.*** and the system will show you suggestions. Select the ***parts*** suggestion or just type it manually and assign alias ***p*** to the joining table.
   You should get the following **Join** clause: ***join {E}.parts p***

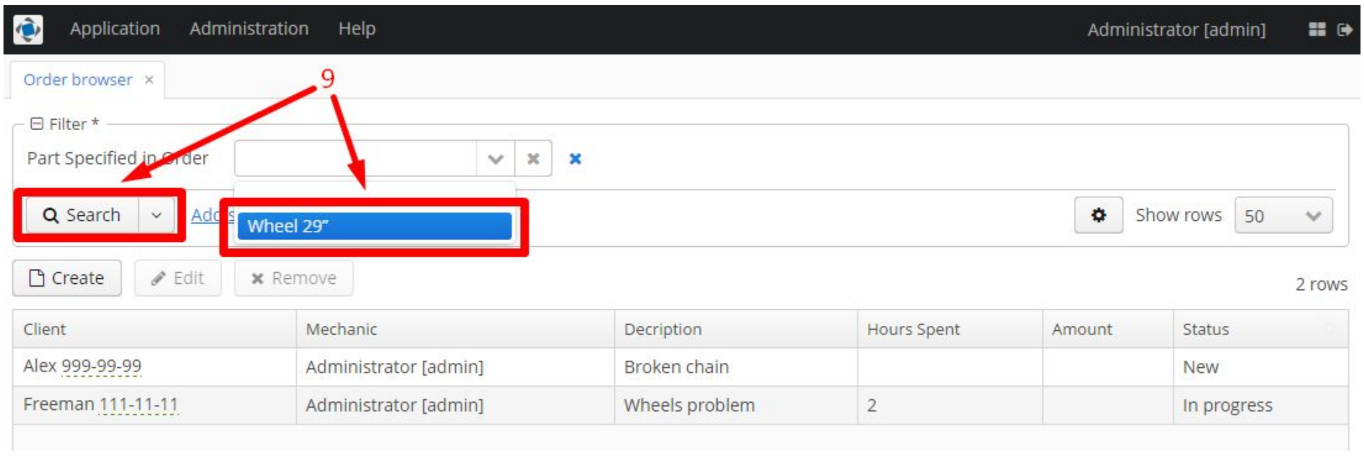


5. Input ***p.id = ?*** in the **Where** field. Question here means parameter, that we will send to the condition from the generic filter UI.
6. Select **Param type**: ***Entity***
7. Set **Entity/Enum**: ***SparePart***
8. Click **OK**


Edit condition

Name	Part Specified in Order
Join	join {E}.parts p
Where	p.id = ?
Param type	Entity
List	☐
Entity/Enum	Spare Part (workshop$SparePart)
Param Where	
Param View	

9. Specify the ***Wheel 29’’*** value in your custom filter and click **Search** to see the result list

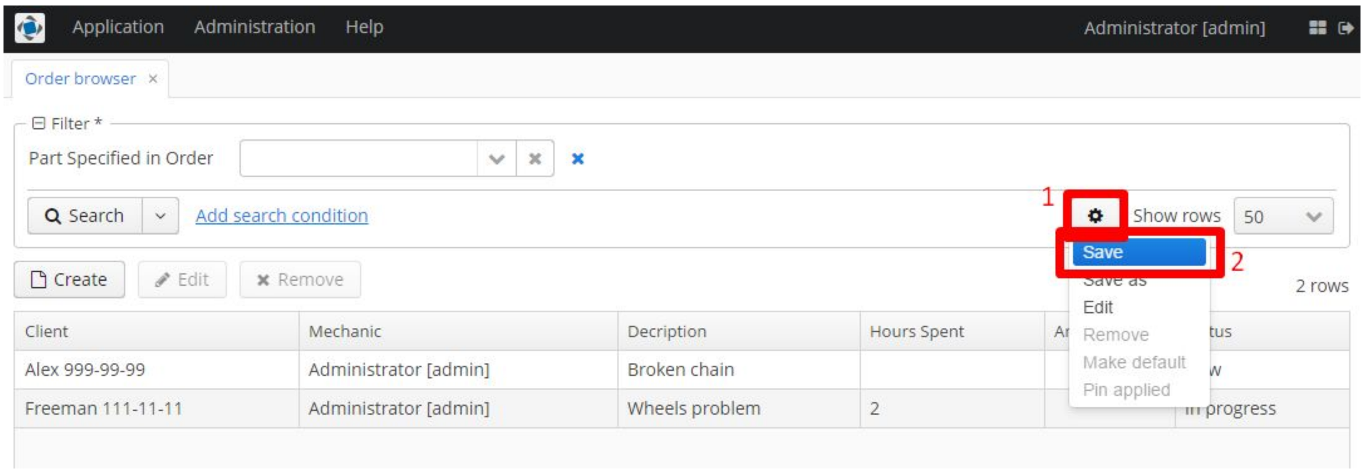


Client	Mechanic	Decription	Hours Spent	Amount	Status
Alex 999-99-99	Administrator [admin]	Broken chain			New
Freeman 111-11-11	Administrator [admin]	Wheels problem	2		In progress

5.1.3 Reusing Filters

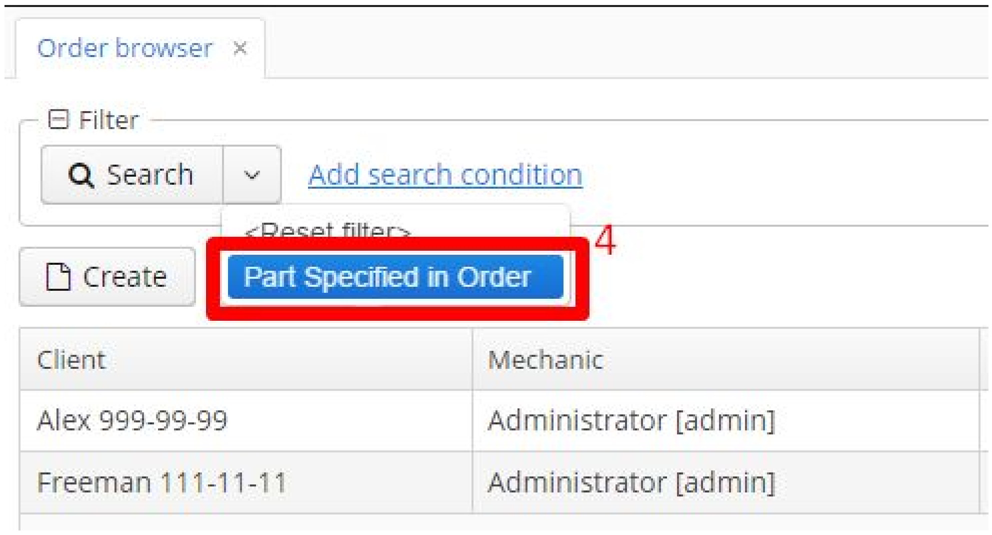
Obviously, there are a number of filters, that end-users reuse frequently. For that purpose you can save filters you create. Let’s save the “part specified in order” filter we have just created.

1. Click the **cogwheel button** on the right hand side of the filter component
2. Click **Save**



Client	Mechanic	Decription	Hours Spent	Ar	tus
Alex 999-99-99	Administrator [admin]	Broken chain			w
Freeman 111-11-11	Administrator [admin]	Wheels problem	2		in progress

3. Input **Filter name**: ***Part specified in order***
4. Now you can access your filter by clicking the **down arrow button**, placed next to the Search one. Apply the saved filter.

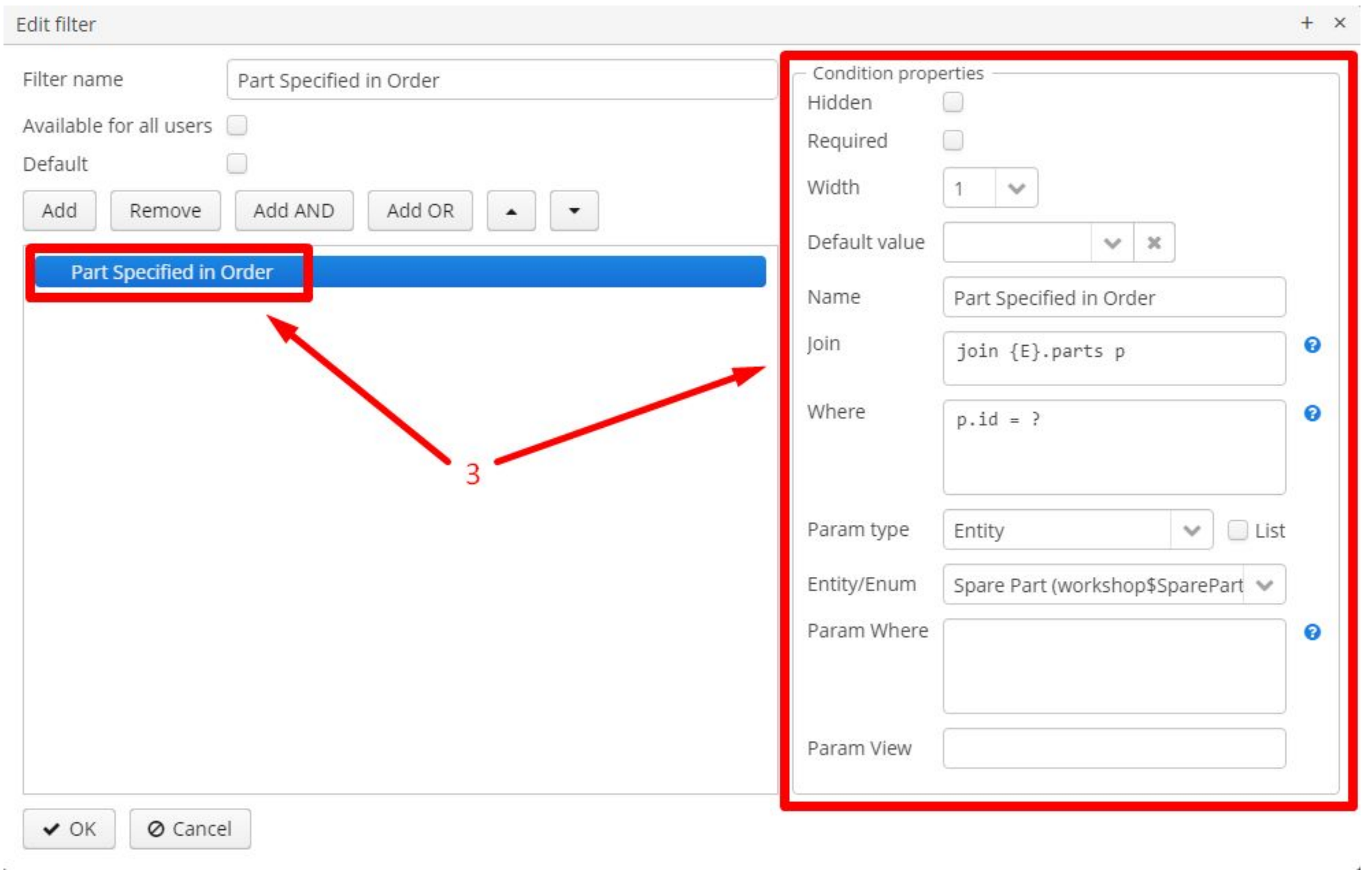


Client	Mechanic
Alex 999-99-99	Administrator [admin]
Freeman 111-11-11	Administrator [admin]

5. Applying a saved filter you can edit, remove it or make selected filter as default one by clicking the **cogwheel button**. Default filters appear in the screen automatically after opening.

5.1.4 Editing Saved Filters and Grouping Conditions

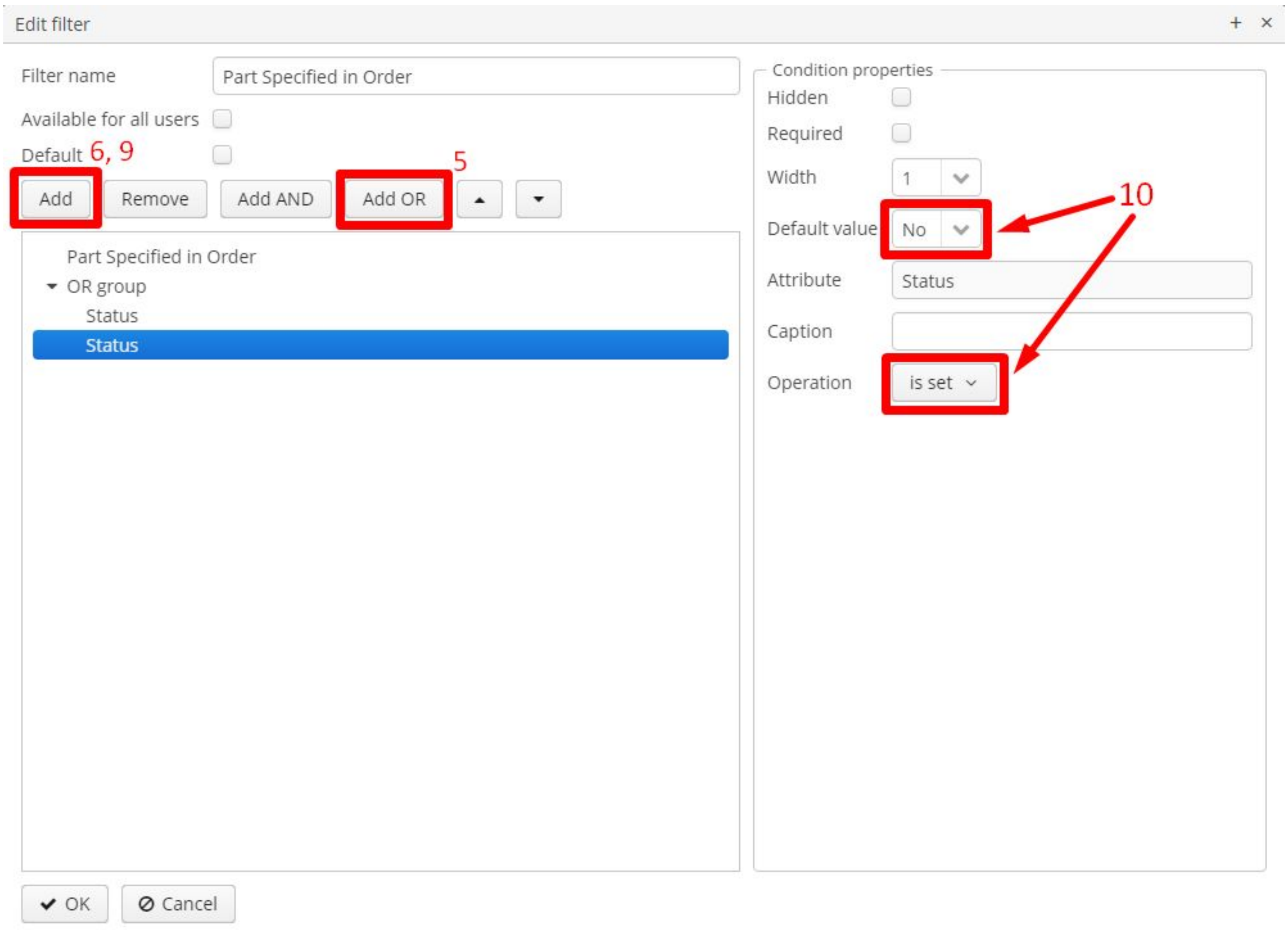
1. Open the saved filter
2. Click the **cogwheel button — Edit**
3. Selecting conditions users are able to edit them


Edit filter

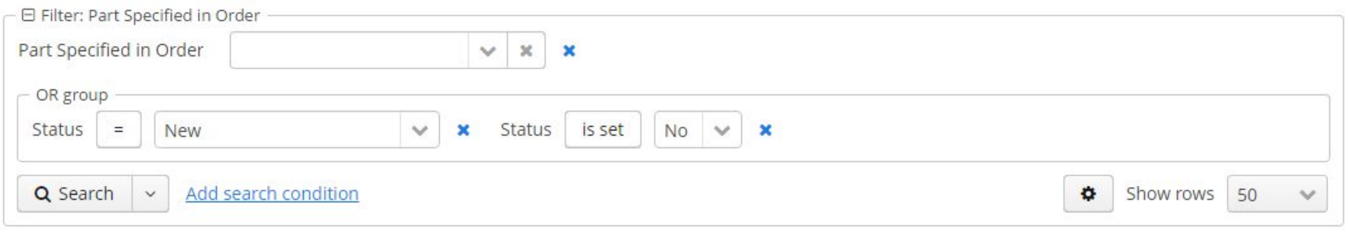
Filter name: Part Specified in Order
Available for all users: ☐
Default: ☐

Condition properties
Hidden: ☐
Required: ☐
Width: 1
Default value:
Name: Part Specified in Order
Join:
```
join {E}.parts p
```
Where:
```
p.id = ?
```
Param type: Entity ☐ List
Entity/Enum: Spare Part (workshop$SparePart
Param Where:
Param View:

4. Also there is an ability to additional conditions and group them using OR and AND logical operators. Let's add one more condition. For example, we want to control our stock of spare parts, so, we would like to know how many parts of some particular type we need for the new or unstatused orders. Logical expression we are going to build is the following:
   Part Specified in Order = ? AND (*Status* = NEW OR *Status* is not set).
5. Click the **Add OR** button and check that the **OR group** item is selected in the conditions list
6. Click **Add**
7. Select the **Status** attribute
8. Set **Default value:** ***New***
9. Select the **OR group** item in the conditions list and add one more condition for the Status attribute
10. Change Operation to **[is set]** one and specify **No** in the **Default value** field



11. Click **OK**
12. Now we can see our complex filter



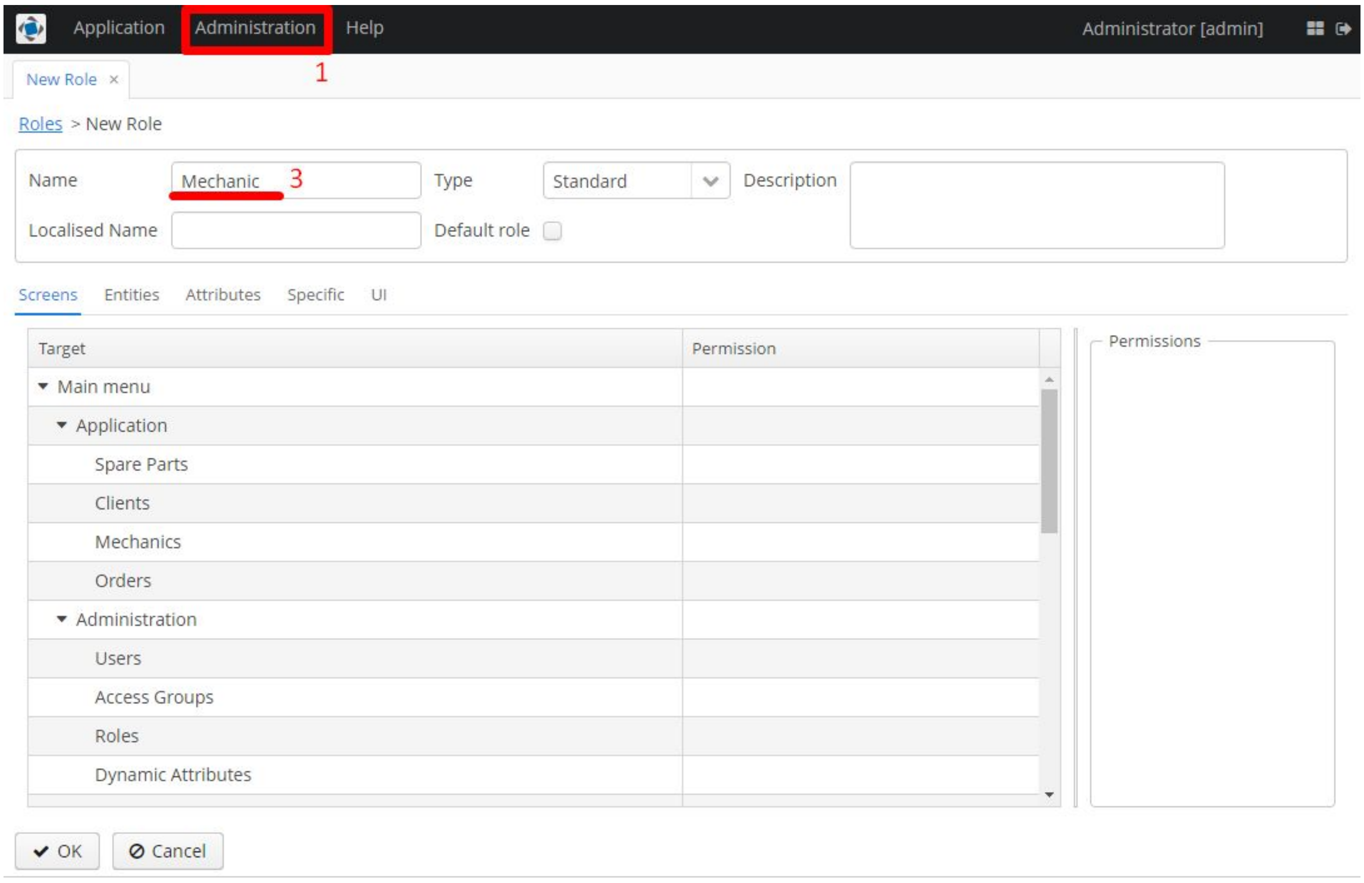
13. Editing filters you can share them between users, set them as default screen filter, hide some conditions or groups of conditions. More information on generic filter component can be found here.

5.2 Security Setup

The platform has built-in functionality to manage users and access rights. This functionality is available from the **Administration** menu. The CUBA platform security model is role-based and controls CRUD permissions for entities, attributes, menu items and screen components and supports custom access restrictions. All security settings can be configured at runtime. There is also an additional facility to control row level access.

We need a role for mechanics in our application. A Mechanic will be able to modify an order and specify the number of hours they spent, and add or remove spare parts. The Mechanic role will have limited administrative functions. Only admin will be allowed to create orders, clients and spare parts.

1. Open **Administration — Roles** from the menu
2. Click **Create**
3. Set **Name**: ***Mechanic***



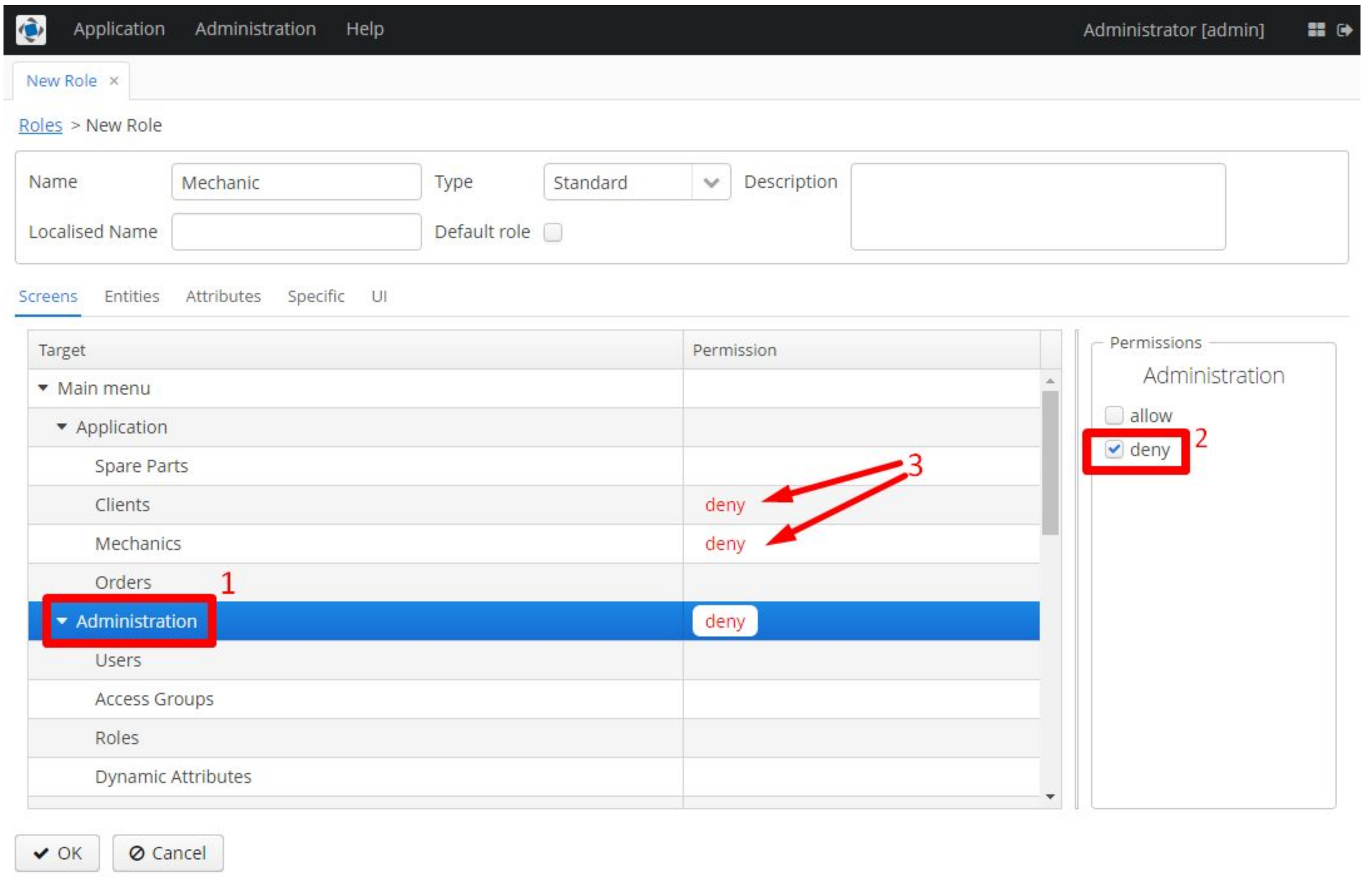
Name: Mechanic
Type: Standard
Description:
Localised Name:
Default role: ☐

Target	Permission
Main menu	
Application	
Spare Parts	
Clients	
Mechanics	
Orders	
Administration	
Users	
Access Groups	
Roles	
Dynamic Attributes	

5.2.1 Screen Permissions

We want to restrict access to the Administration screens for all Mechanic users, so let's forbid the Administration menu item. Also, mechanics don't need access to the mechanics and clients browsers, let's forbid the corresponding screens too.

1. Select the **Administration** row in the Screens table
2. Select **deny**
3. Similarly deny access for **Clients** and **Mechanics**

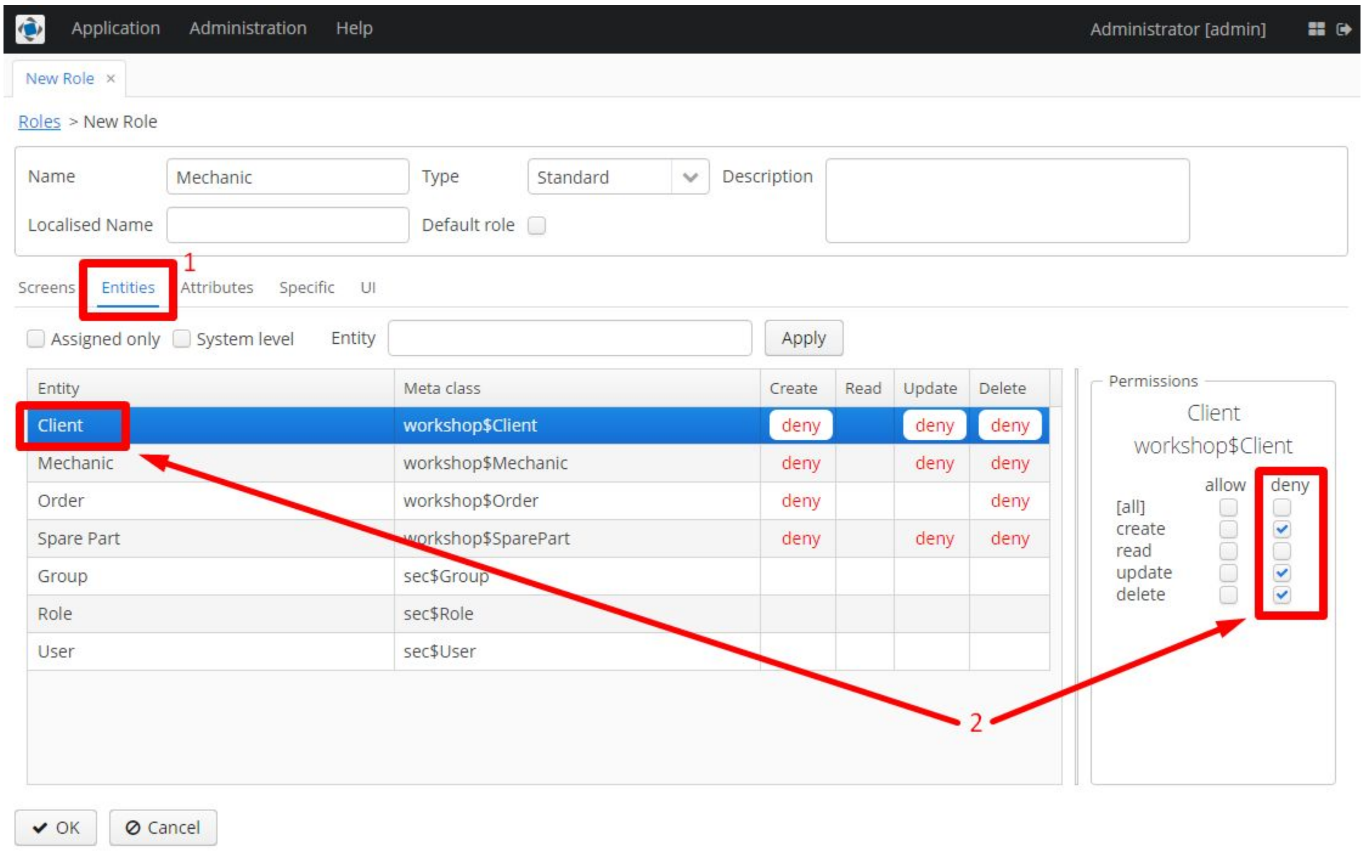


Target	Permission
Main menu	
Application	
Spare Parts	
Clients	deny
Mechanics	deny
Orders	
Administration	deny
Users	
Access Groups	
Roles	
Dynamic Attributes	

5.2.2 CRUD Permissions

As it has already been mentioned we want a mechanic only allow read access to clients, mechanics and parts. Also a mechanic should not be able to create or delete an order in the system. Only update it's status, hours spent and manipulate with parts in the order.

1. Open the **Entities** tab
2. Select the **Client** entity and forbid **create**, **update** and **delete** operations



Entity	Meta class	Create	Read	Update	Delete
Client	workshop$Client	deny		deny	deny
Mechanic	workshop$Mechanic	deny		deny	deny
Order	workshop$Order	deny			deny
Spare Part	workshop$SparePart	deny		deny	deny
Group	sec$Group				
Role	sec$Role				
User	sec$User				

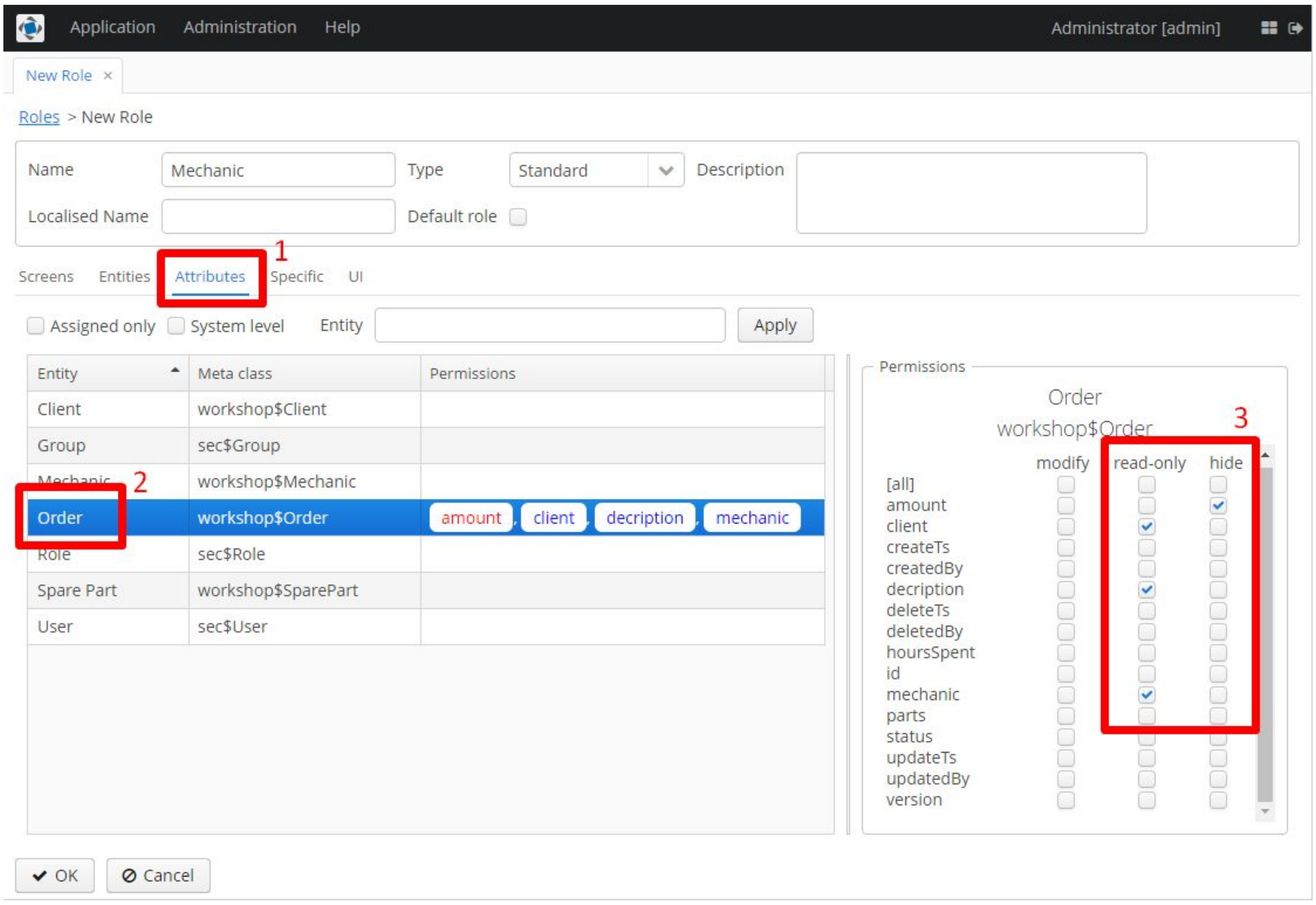
	allow	deny
[all]	☐	☐
create	☐	✓
read	☐	☐
update	☐	✓
delete	☐	✓

3. Repeat the second step for the **Mechanic** and **SparePart** entities
4. Forbid only **create** and **delete** operations for the **Order** entity

5.2.3 Attribute Permissions

On attribute level we don't want to allow a mechanic to set values for the client, mechanic and description attributes; also we would like to hide the amount field from them.

1. Open the **Attributes** tab
2. Select the **Order** row and
3. Tick **read-only** for **client**, **mechanic** and **description**; tick **hide** for the **amount** attribute



Name	Mechanic	Type	Standard	Description	
Localised Name		Default role	☐		

☐ Assigned only ☐ System level Entity

Entity	Meta class	Permissions
Client	workshop$Client	
Group	sec$Group	
Mechanic	workshop$Mechanic	
Order	workshop$Order	amount, client, decription, mechanic
Role	sec$Role	
Spare Part	workshop$SparePart	
User	sec$User	

Permissions

Order
workshop$Order

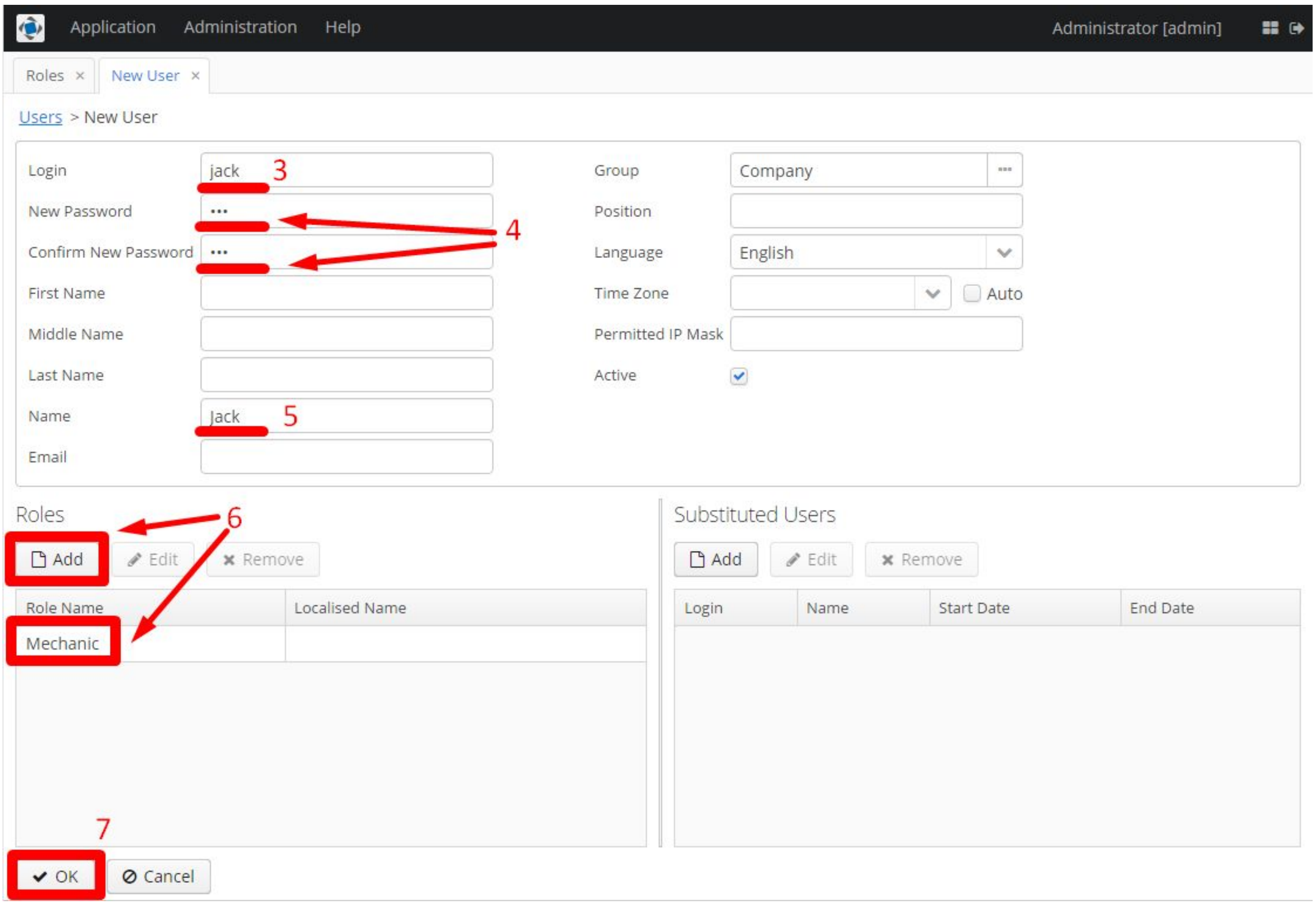
	modify	read-only	hide
[all]	☐	☐	☐
amount	☐	☐	✓
client	☐	✓	☐
createTs	☐	☐	☐
createdBy	☐	☐	☐
decription	☐	✓	☐
deleteTs	☐	☐	☐
deletedBy	☐	☐	☐
hoursSpent	☐	☐	☐
id	☐	☐	☐
mechanic	☐	✓	☐
parts	☐	☐	☐
status	☐	☐	☐
updateTs	☐	☐	☐
updatedBy	☐	☐	☐
version	☐	☐	☐

4. Click **OK** to save the role

5.2.4 Role-based Security in Action

Let’s apply the mechanic role to a new user and take a look on the application from the mechanic’s point of view.

1. Open **Administration — Users**
2. Click **Create**
3. Set **Login**: ***jack***
4. Set **password** and **password confirmation**: ***qwe***
5. Set **Name**: ***Jack***
6. Add the **Mechanic** role to user **Roles**
7. Click **OK** to save the user



Login	jack
New Password	•••
Confirm New Password	•••
First Name	
Middle Name	
Last Name	
Name	Jack
Email	
Group	Company
Position	
Language	English
Time Zone	
Auto	☐
Permitted IP Mask	
Active	✓

Role Name	Localised Name
Mechanic	

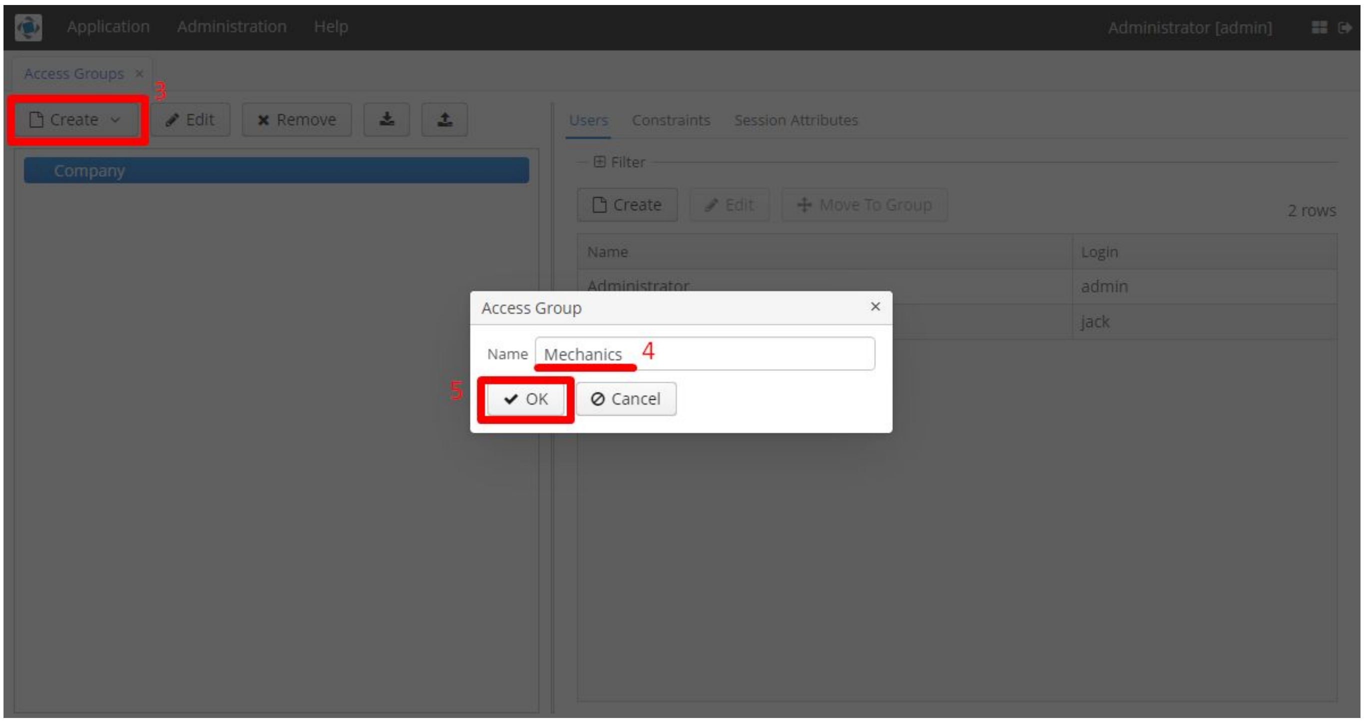
Login	Name	Start Date	End Date

8. Let's exit the application by clicking the **top-right door button** of the screen
9. Login under the new user Login:***jack***, Password:***qwe***
10. The Administration menu item is hidden as well as Mechanics and Clients browsers;
    The Spare Parts browser is available in read-only mode;
    The Orders browser allows us only editing of existing records;
    The Order editor allows only changing the Hours spent and Status fields, as well as adding parts to an order; the amount field is hidden from our user.

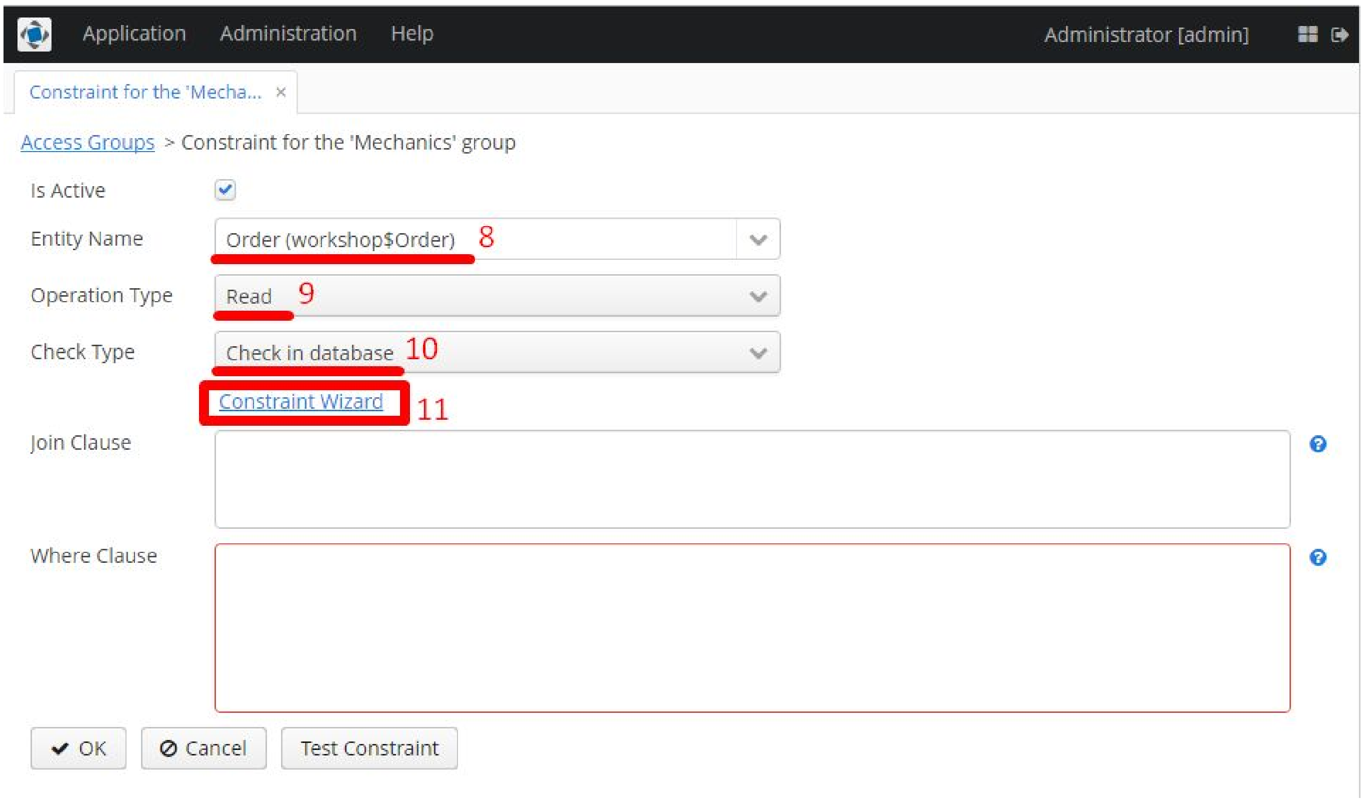
5.2.5 Row-level Security

The CUBA Platform Security subsystem allows you to control and restrict access on the level of records using the **access group** mechanism. For example, we want to restrict access of our mechanics to see only orders that were assigned to them.

1. **Log out** from the system and **login** under administrator (**Login**: ***admin***; **Password**: ***admin***)
2. Open **Administration — Access Groups** from the menu. The groups have hierarchical structure, where each element defines a set of constraints, allowing controlling access to individual entity instances (at row level).
3. Click **Create — New**
4. Set **Name:** ***Mechanics***
5. Click **OK**

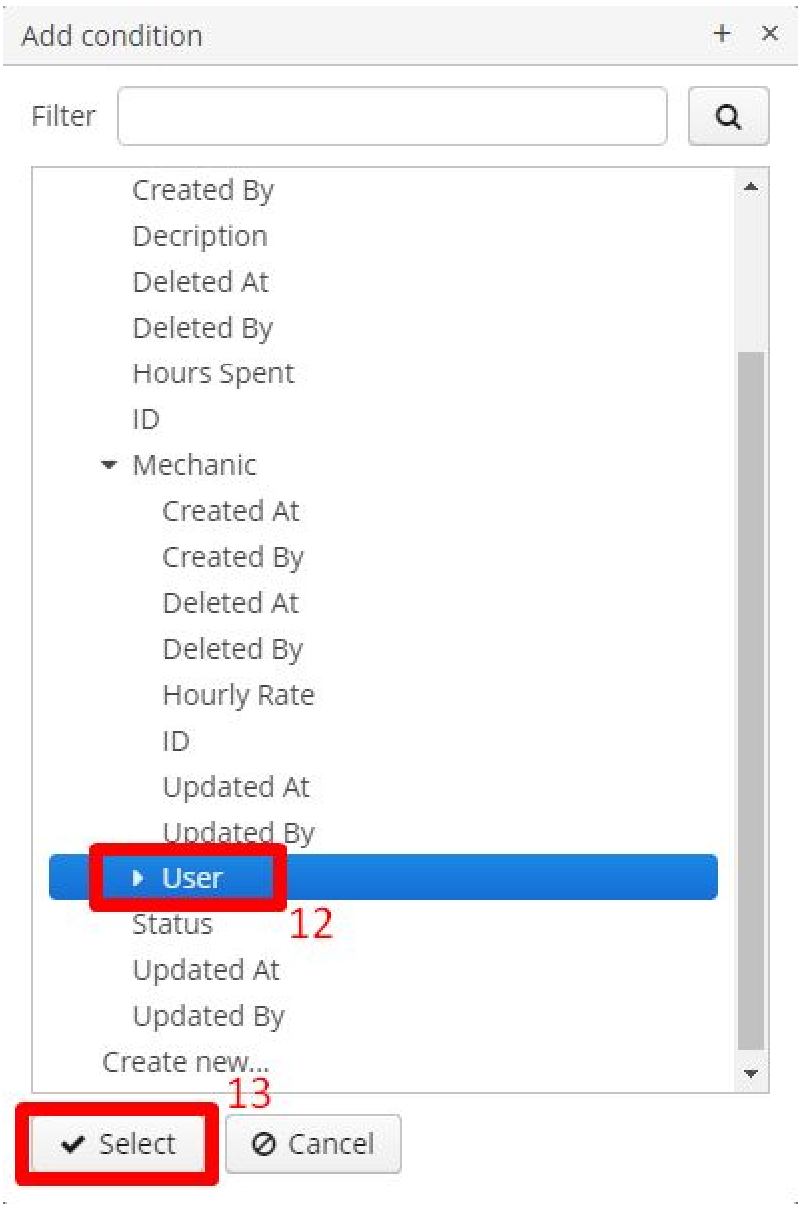


6. Open the **Constraints** tab for the newly created group
7. Click **Create** in the **Constraints** tab
8. Set **Entity Name**: ***Order***
9. We can define what operation type we want to assign restriction to. Keep **Operation Type:** ***Read***
10. For some operation types it is not possible to perform check on the database level, e.g. delete and update operations. For this reason there are three **Check Types** implemented in the platform: *Check in database* (using JPQL), *Check in memory* (using Groovy script; the only option for Create, Update, Delete, All operation types) and *Check in database and in memory* (using both JPQL and Groovy). In our example we will use the ***Check in database*** Check Type
11. For the *Check in database* Check Type you can specify a JPQL query in the same way as we did for custom conditions in the generic filter. Click the **Constraint Wizard** link

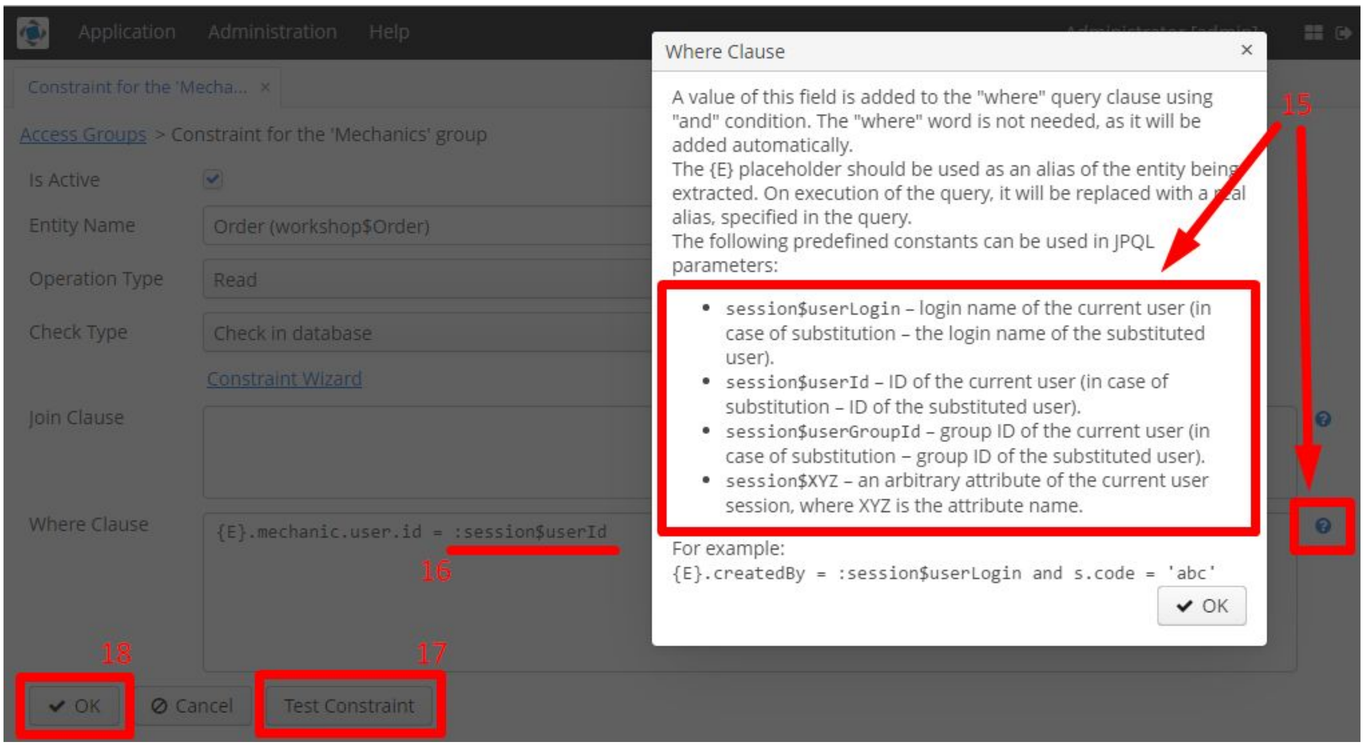

Application
Administration
Help
Administrator [admin]
Constraint for the 'Mecha... ×
Access Groups > Constraint for the 'Mechanics' group

Is Active: ✓
Entity Name: Order (workshop$Order) 8
Operation Type: Read 9
Check Type: Check in database 10

12. Expand the **Mechanic** field and select the child **User** attribute
13. Click **Select**



14. Click **OK** in the Define query window. The platform has generated the where clause as *{E}.mechanic.user.id* = *'NULL'*. We will need to tune it a bit.
15. Click the question mark located in the top-right corner of the Where Clause filed to see what predefined constants can be used here
16. We need ID of the current user to compare with the mechanic's user id. So, change *'NULL'* to :*session$userId*. Finally you should have the following code:
    {E}.mechanic.user.id = :session$userId
17. Click **Test Constraint**
18. If test shows that constraint is valid, then click **OK**


Where Clause

A value of this field is added to the "where" query clause using "and" condition. The "where" word is not needed, as it will be added automatically.
The {E} placeholder should be used as an alias of the entity being extracted. On execution of the query, it will be replaced with a real alias, specified in the query.
The following predefined constants can be used in JPQL parameters:

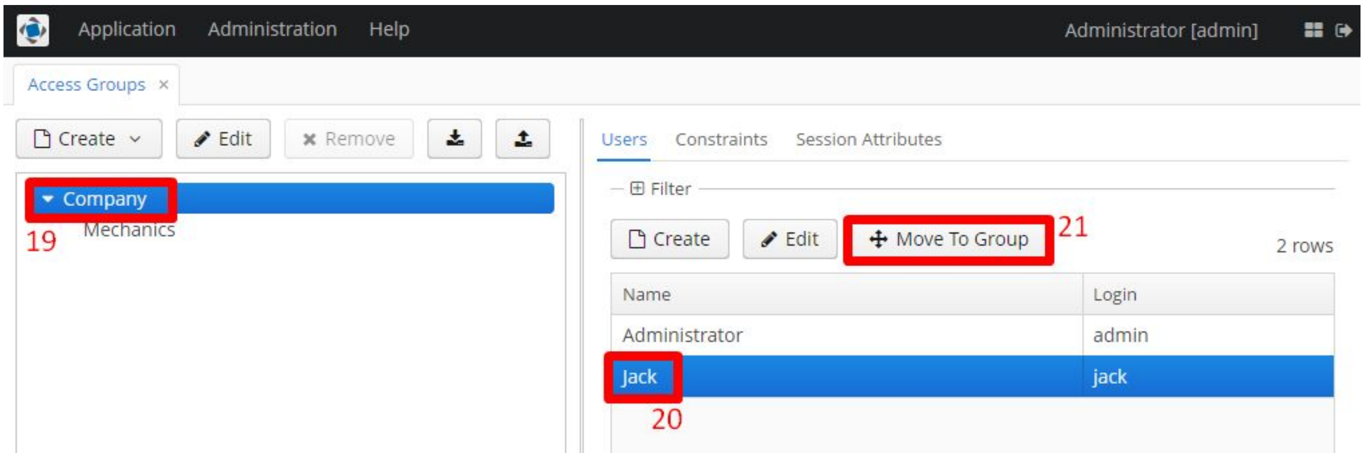
- session$userLogin – login name of the current user (in case of substitution – the login name of the substituted user).
- session$userId – ID of the current user (in case of substitution – ID of the substituted user).
- session$userGroupId – group ID of the current user (in case of substitution – group ID of the substituted user).
- session$XYZ – an arbitrary attribute of the current user session, where XYZ is the attribute name.

For example:
`{E}.createdBy = :session$userLogin and s.code = 'abc'`

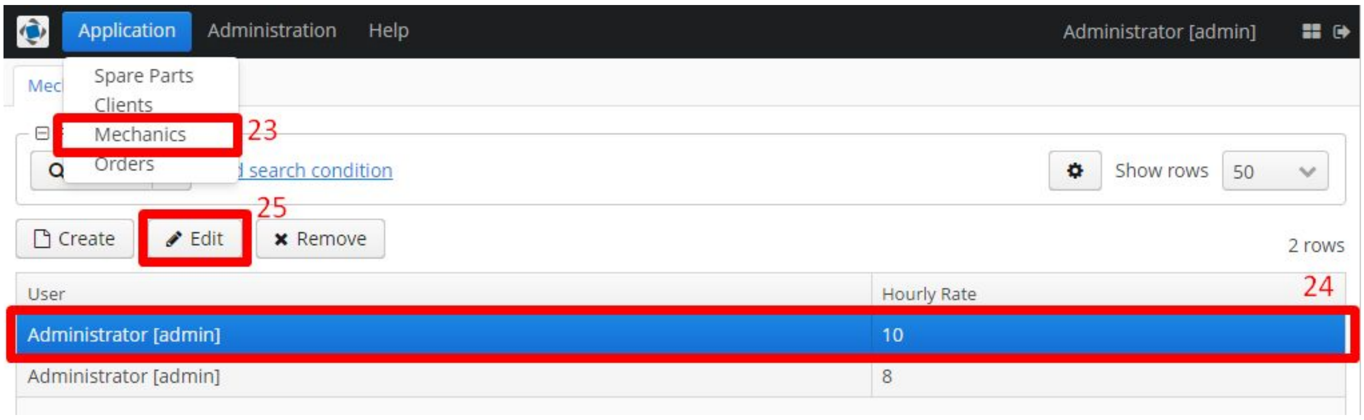
Is Active	✓
Entity Name	Order (workshop$Order)
Operation Type	Read
Check Type	Check in database
Join Clause	
Where Clause	`{E}.mechanic.user.id = :session$userId`

Application
Administration
Help
Constraint for the 'Mecha...
Access Groups > Constraint for the 'Mechanics' group
Constraint Wizard
OK
Cancel
Test Constraint
15
16
17
18

19. Select the **Company** group on the **Access Groups** screen
20. Select ***Jack*** in the **Users** tab
21. Click **Move To Group**


Name	Login
Administrator	admin
Jack	jack

22. Select the **Mechanics** group and click **Select**
23. Let's now assign one of our mechanics with the new user. Go to **Application - Mechanics** in the main menu
24. Select the **mechanic with hourly rate of 10**
25. Click **Edit**



User	Hourly Rate
Administrator [admin]	10
Administrator [admin]	8

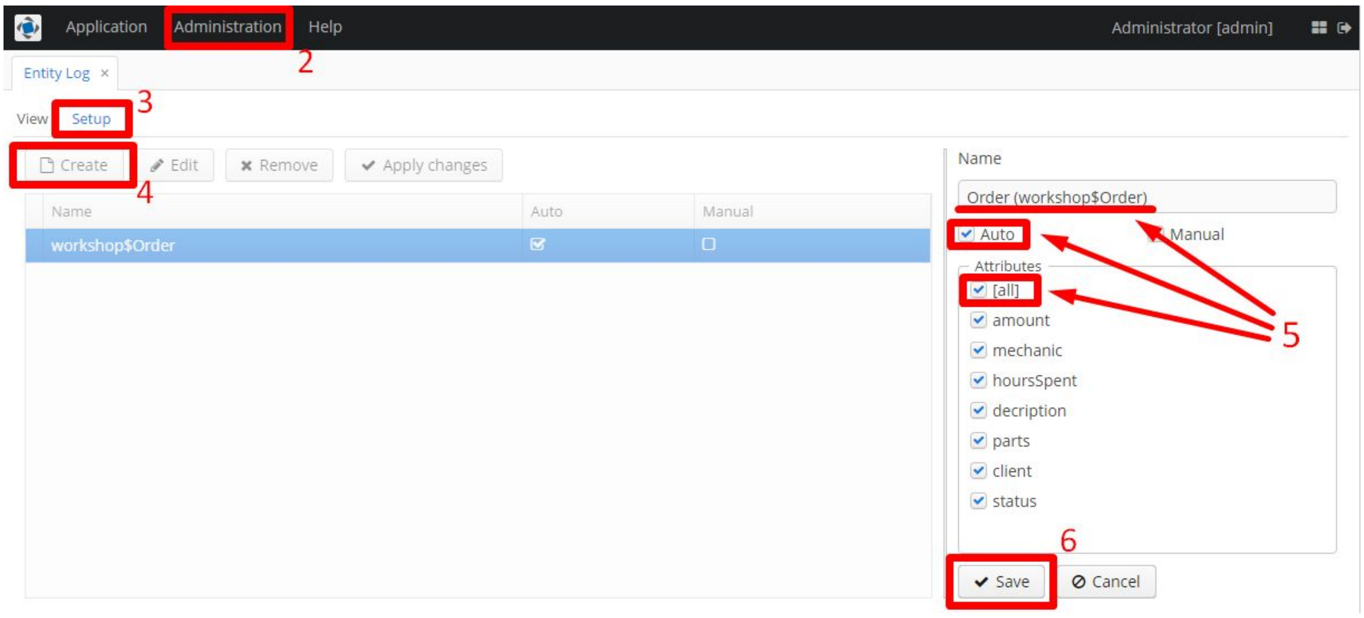
26. Specify the ***Jack*** user for this record and click **OK**
27. Now our mechanic is associated with the jack user. Re-login into the application using Jack's credentials: **Login**: ***jack***; **Password**: ***qwe***
28. Open the Orders browser (**Application — Orders**). Now we can only see the orders assigned to Jack

5.3 Audit

There is a requirement in our functional specification to track critical data changes. CUBA Platform has a built-in mechanism to track entity changes, which you can configure to track operations with critical system data.

Keeping track of our application, it happens when one day someone has accidentally erased the order description. It is not appropriate to call the client on the phone, apologize and ask them to repeat the what needs to be done. Let's see how this can be avoided.

1. Re-login into the application as administrator (**Login**: ***admin***; **Password**: ***admin***)
2. Open **Administration — Entity Log** from the menu
3. Go to the **Setup** tab
4. Click **Create**
5. Set **Name**: ***Order***
   Auto: ***true***
   Attributes: *all*
6. Click **Save**



Name	Auto	Manual
workshop$Order	✓	

Name: Order (workshop$Order)

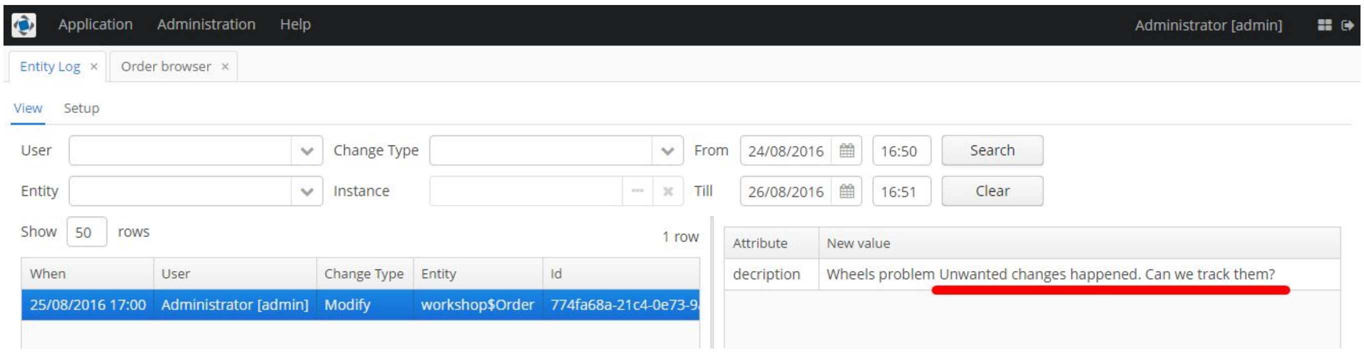
[✓ Auto] Manual

Attributes

- ✓ [all]
- ✓ amount
- ✓ mechanic
- ✓ hoursSpent
- ✓ decription
- ✓ parts
- ✓ client
- ✓ status

Now the system will track all CRUD changes of all the fields of the Order entity. Go to the order browser and change status or description of any record, them get back to the Entity Log screen and you will see who made what change!

I changed the description of one of the orders and see what we have in the entity log:



User:
Change Type:
From: 24/08/2016 16:50
Entity:
Instance:
Till: 26/08/2016 16:51
Show 50 rows
1 row

When	User	Change Type	Entity	Id
25/08/2016 17:00	Administrator [admin]	Modify	workshop$Order	774fa68a-21c4-0e73-9

Attribute	New value
decription	Wheels problem Unwanted changes happened. Can we track them?

5.4 First Launch Conclusion

While spending short time exploring the CUBA Platform and its features we have already finished the following requirements from the functional specification:

1. ~~Store customers with their name, mobile phone and email~~
2. ~~Record information about orders: price for repair and time spent by mechanic~~
3. ~~Keep track of spare parts in stock~~
4. Automatically calculate price based on spare parts used and time elapsed
5. ~~Control security permissions for screens, CRUD operations and records' attributes~~
6. ~~Audit of critical data changes~~
7. API for a mobile client to place an order

Note that we didn't even type any line of source code, everything has been provided by the platform or scaffolded by the Studio.

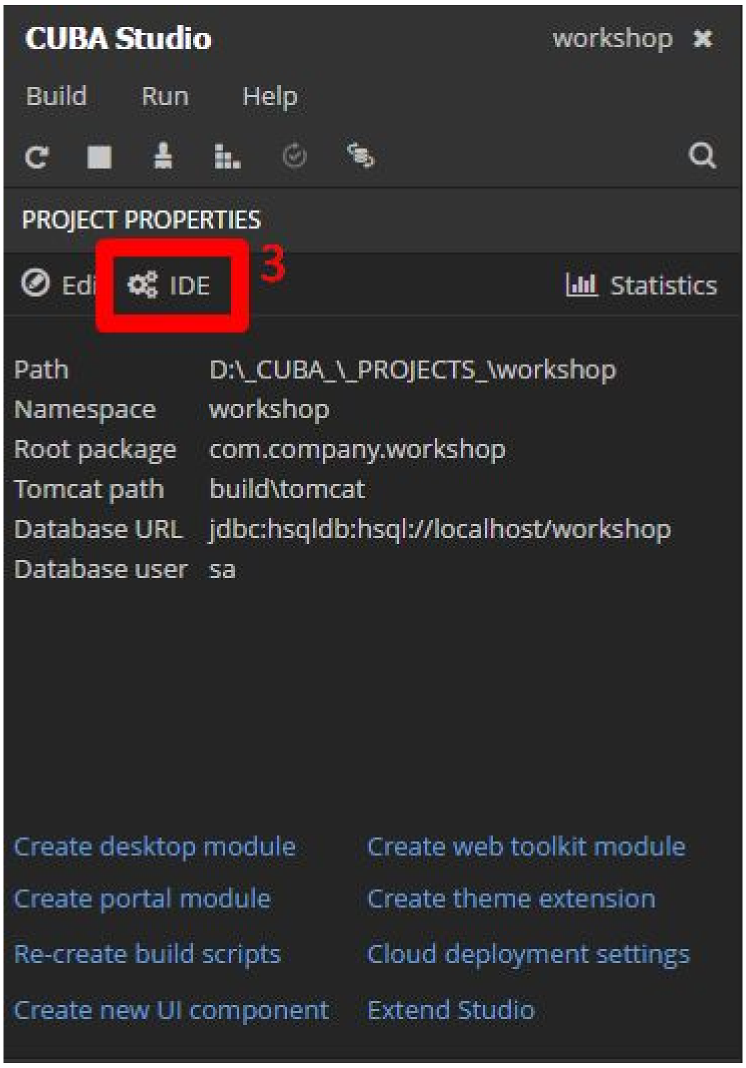
Let's move forward towards business logic.

6 DEVELOPMENT BEYOND CRUD

6.1 Integration with IDE and Project Structure

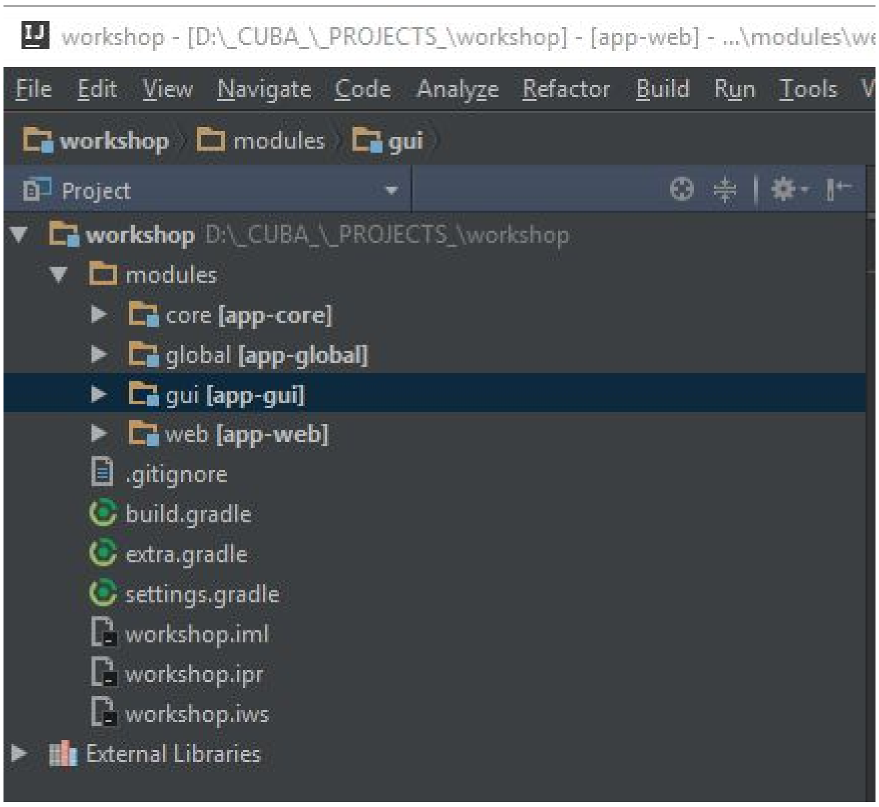
Let’s have a look how our project looks from inside. Keep your application up and running and follow the steps:

1. Launch IntelliJ IDEA. The IDE should be up and running to enable integration with the CUBA Studio. If you don’t have the CUBA Plugin installed, please install it, cause it is used as a communication bridge between the Studio and IDE
2. Go to the Studio and click the **IDE** button in the **Project properties** section. The Studio will generate project files and open the project in the IDE



Path	D:_CUBA__PROJECTS_\workshop
Namespace	workshop
Root package	com.company.workshop
Tomcat path	build\tomcat
Database URL	jdbc:hsqldb:hsql://localhost/workshop
Database user	sa

3. Move to the IDE and press **Alt+1** to see the project structure. By default any project consists of 4 modules: **global**, **core**, **web**, **gui**.
   global - data model classes
   core - middle tier services
   gui - common component interfaces; screens and components for both Web and Desktop clients and
   web - screens and components for Web client



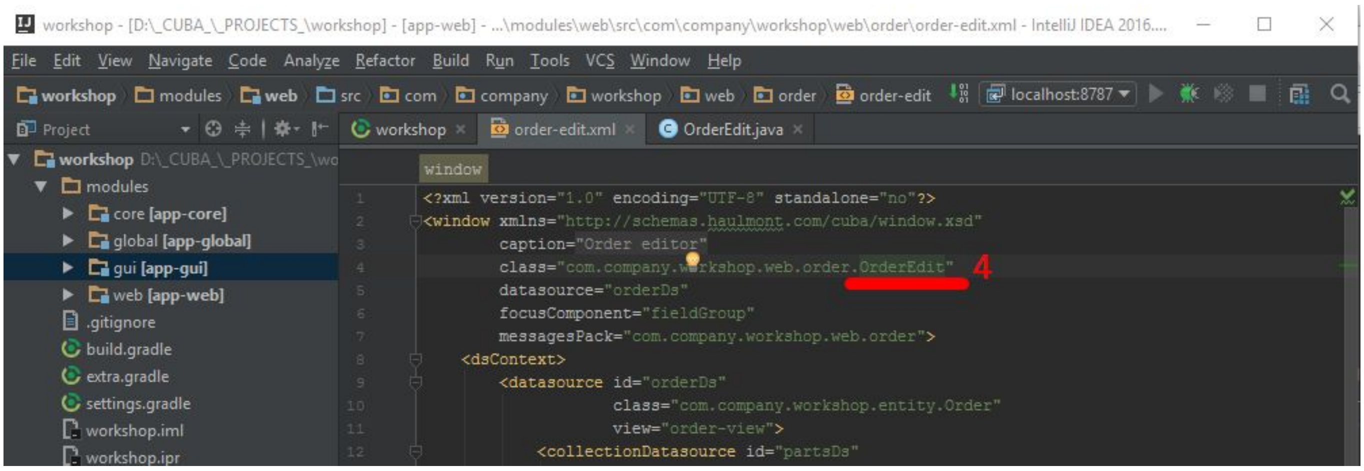
6.2 Customization of Existing Screens and Hotdeploy

In this chapter we will polish our Orders browser and editor by adding logic into controller and changing the user interface.

6.2.1 Initialization of a New Record in Editor

Let's solve the first problem with empty status of the newly creating orders. A new order should be created with the NEW order status pre-set.

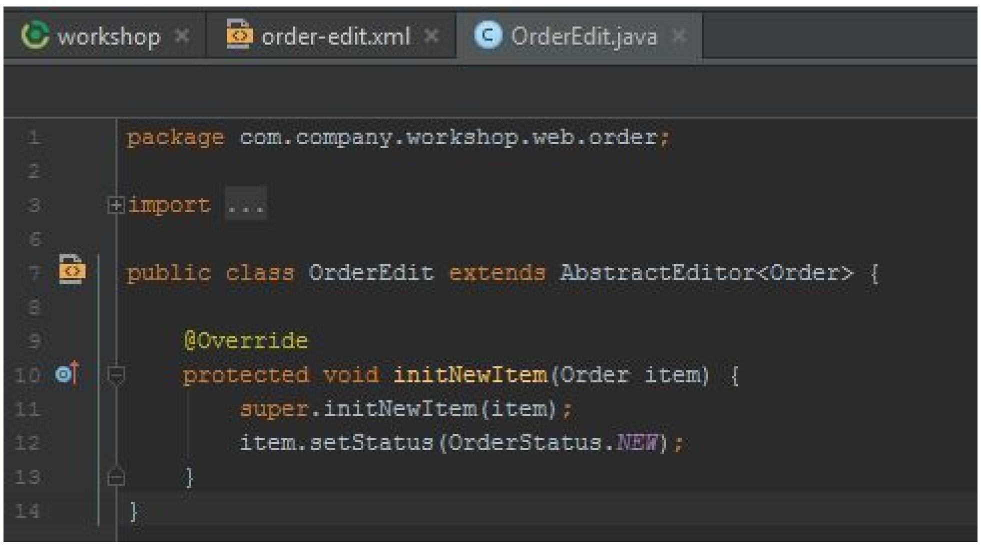
1. Go to the **Screens** section of the navigation panel in the CUBA Studio
2. Select the **order-edit.xml** screen
3. Click the **IDE** button on top of the section. Screen descriptor appears in your IDE. You can make any changes right from the source code, because The Studio and an IDE has two ways synchronization
4. Hold **Ctrl** button and click on **OrderEdit** in class attribute of the XML descriptor to navigate to its implementation


workshop - [D:_CUBA__PROJECTS_\workshop] - [app-web] - ...\modules\web\src\com\company\workshop\web\order\order-edit.xml - IntelliJ IDEA 2016....
4

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<window xmlns="http://schemas.haulmont.com/cuba/window.xsd"
        caption="Order editor"
        class="com.company.workshop.web.order.OrderEdit"
        datasource="orderDs"
        focusComponent="fieldGroup"
        messagesPack="com.company.workshop.web.order">
    <dsContext>
        <datasource id="orderDs"
                    class="com.company.workshop.entity.Order"
                    view="order-view">
            <collectionDatasource id="partsDs"
```

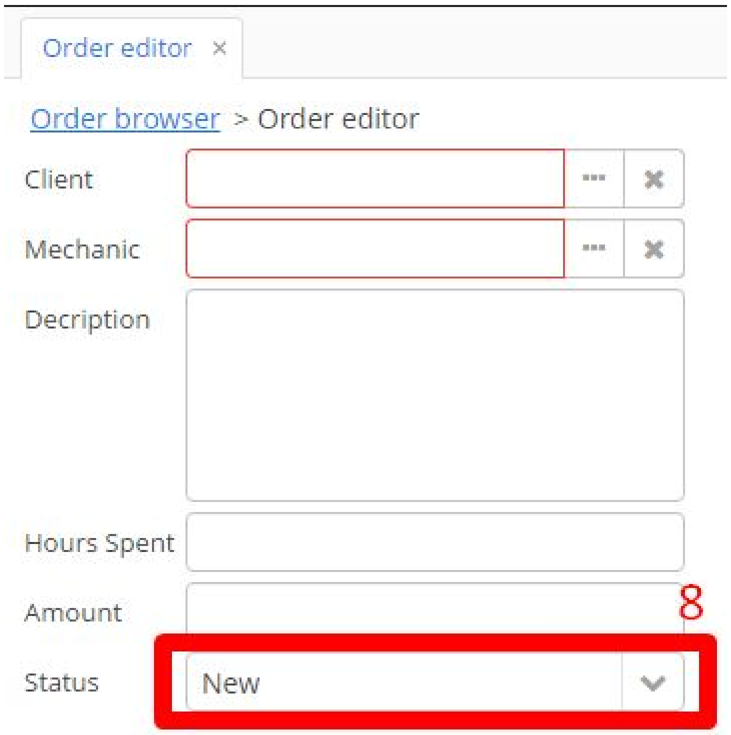
5. Override method ***initNewItem*** and set status ***OrderStatus.NEW*** to the passed order:

```
public class OrderEdit extends AbstractEditor<Order> {
        @Override
        protected void initNewItem(Order item) {
                super.initNewItem(item);
                item.setStatus(OrderStatus.NEW);
        }
}
```



```
package com.company.workshop.web.order;

import ...

public class OrderEdit extends AbstractEditor<Order> {

    @Override
    protected void initNewItem(Order item) {
        super.initNewItem(item);
        item.setStatus(OrderStatus.NEW);
    }
}
```

6. We haven’t stopped our application and it is still up and running in the browser. Open/Reopen **Application — Orders** screen
7. Click **Create**
8. We see our changes, although we haven’t restarted the server. The CUBA Studio automatically detects and the hot-deploys changes, except for the data model, which saves a lot of time while UI development and business logic development


Order editor ×

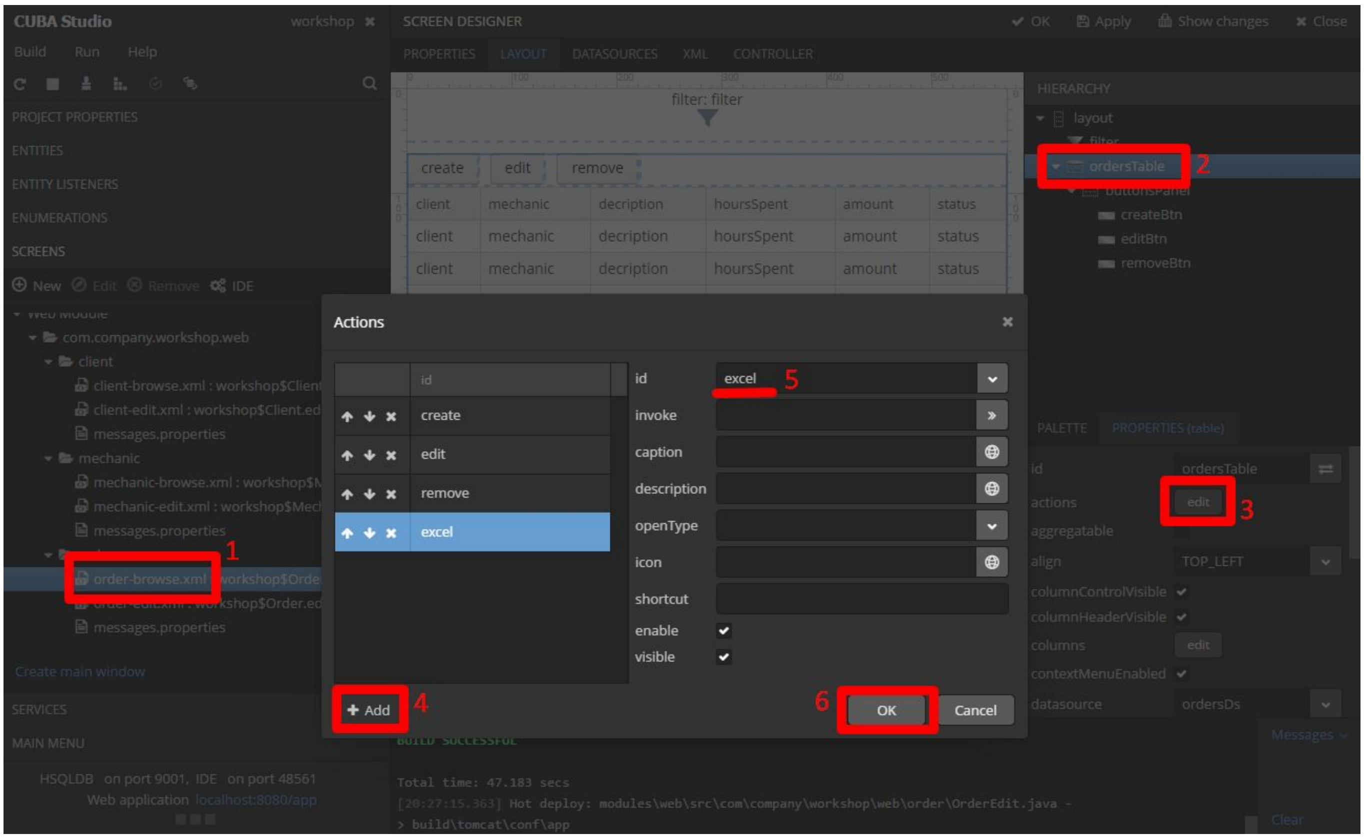
Order browser > Order editor

Client	
Mechanic	
Decription	
Hours Spent	
Amount	
Status	New

6.2.2 Adding Standard Excel Action to the Orders Browser

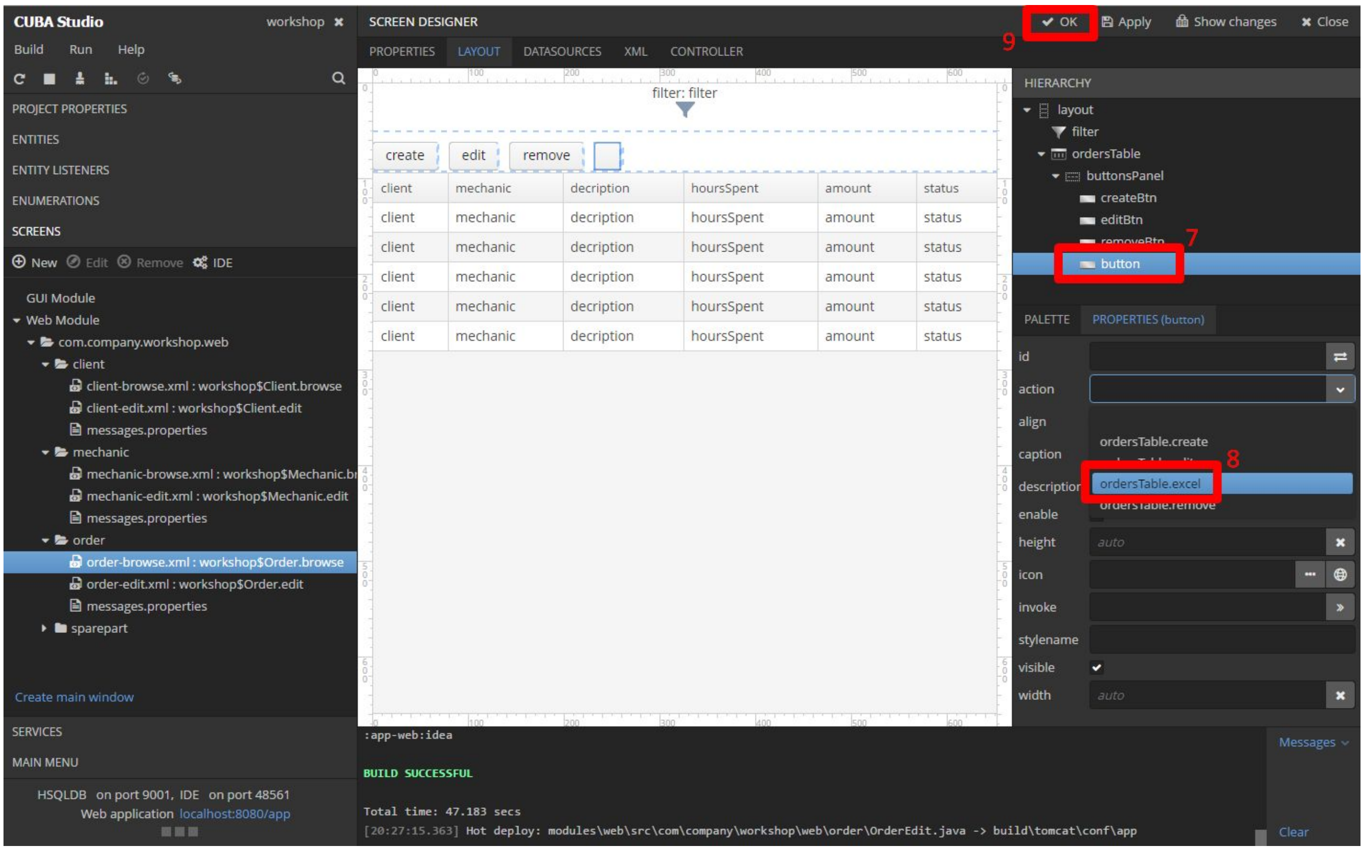
The standard screens contain **Create**, **Edit**, and **Remove** actions by default. Let’s add an action to export the order list to **Excel**, which is also a standard action you can use out of the box. You can follow the link to learn more about standard actions.

1. Open **order-browse.xml** screen in the Studio
2. Select table component, go to properties panel
3. Click the **edit** button in the actions property
4. **Add** a new action row to the list
5. Specify id as ***excel*** for this action
6. Click **OK**



```
BUILD SUCCESSFUL

Total time: 47.183 secs
[20:27:15.363] Hot deploy: modules\web\src\com\company\workshop\web\order\OrderEdit.java -
> build\tomcat\conf\app
```

7. Add a new button to the button panel (**drug and drop** it from the components palette into the hierarchy of components)
8. Select ***ordersTable.excel*** action for button using properties panel
9. Save the screen by clicking **OK** in the top-right corner



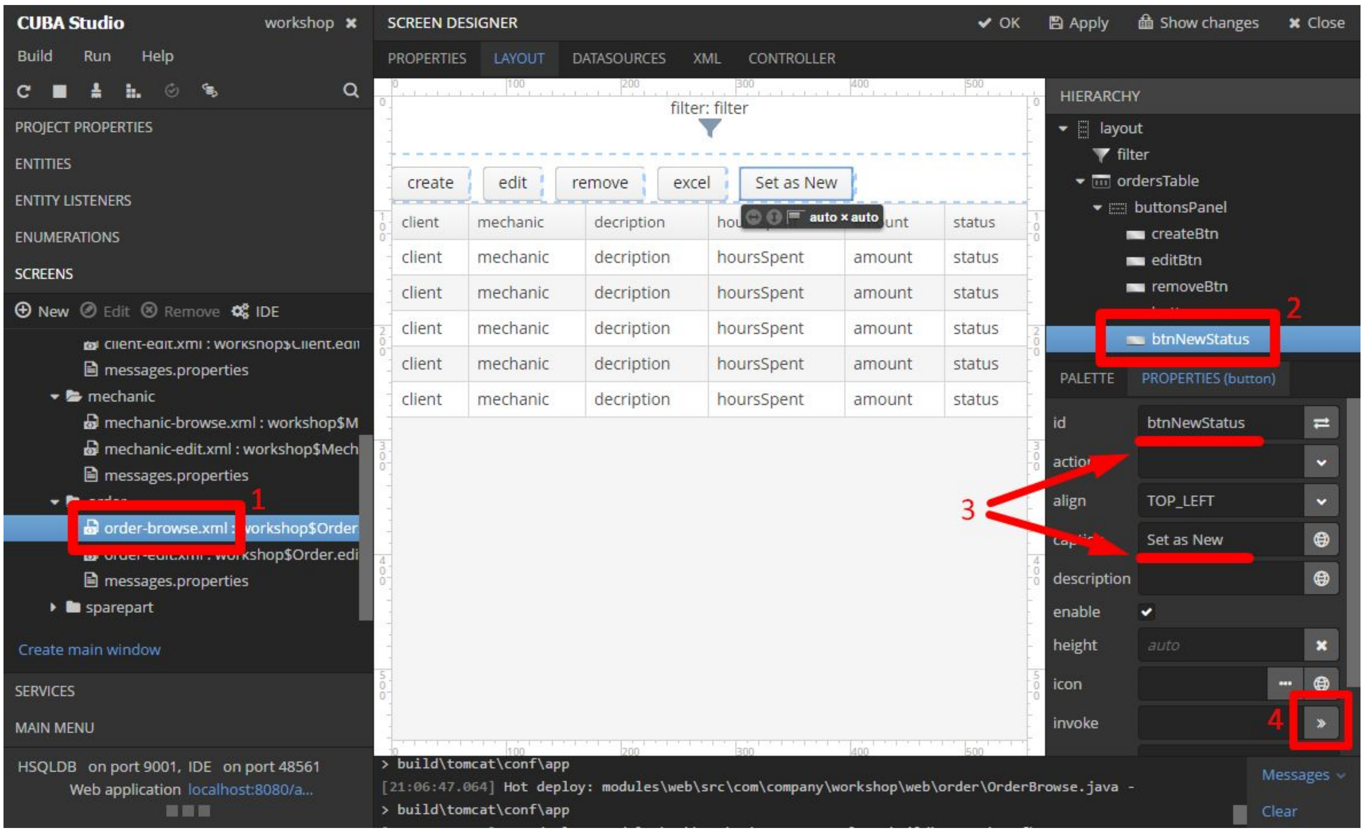
```
:app-web:idea

BUILD SUCCESSFUL

Total time: 47.183 secs
[20:27:15.363] Hot deploy: modules\web\src\com\company\workshop\web\order\OrderEdit.java -> build\tomcat\conf\app
```

10. Let's use magic of hotdeploy once again. **Open/Reopen** the Orders screen
11. Click **Excel** to export your orders to an xls file

6.2.3 Adding Custom Behaviour

Our mechanics are yelling that it's too annoying to go to the Order editor every time they want to change the order status. They would like to click a button and set the corresponding status from the browser.

1. Open **order-browse.xml** screen in the Studio
2. Add a new button to the button panel (**drug and drop** it into the hierarchy of components)
3. Set the following properties for the button
   id: ***btnNewStatus***
   caption: ***Set as New***
4. Click the **[>>]** button on the right side of the invoke field



id	btnNewStatus
action	
align	TOP_LEFT
caption	Set as New
description	
enable	✓
height	auto
icon	
invoke	

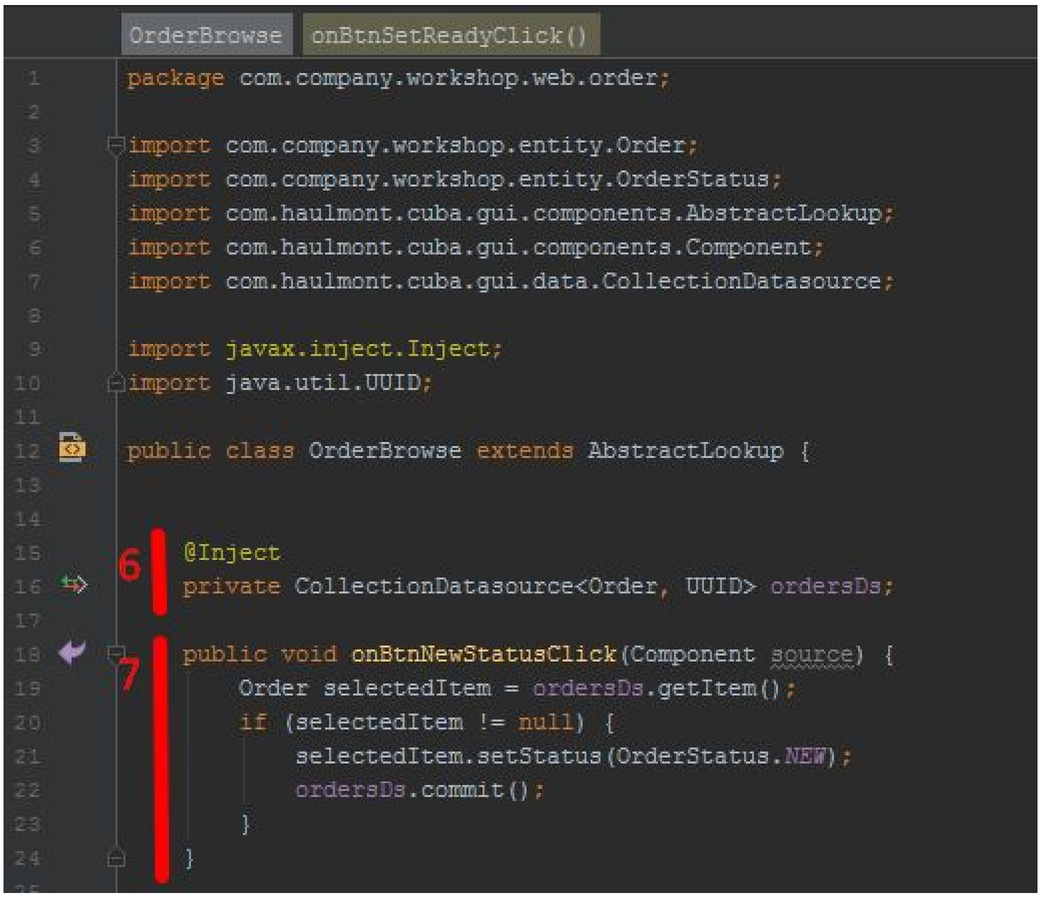
```
> build\tomcat\conf\app
[21:06:47.064] Hot deploy: modules\web\src\com\company\workshop\web\order\OrderBrowse.java -
> build\tomcat\conf\app
```

5. The Studio will generate a method, that will be called on button click. Press **Ctrl+I** to save the changes and open the screen controller in your IDE
6. CUBA extends the standard spring injection mechanism with ability to inject CUBA related infrastructure and UI components. We will need to inject the datasource from our screen:

```
@Inject
private CollectionDatasource<Order, UUID> ordersDs;
```

7. Let's implement the ***onBtnNewStatusClick*** method, that was created by the Studio:

```
public void onBtnNewStatusClick(Component source) {
   Order selectedItem = ordersDs.getItem();
   if (selectedItem != null) {
       selectedItem.setStatus(OrderStatus.NEW);
       ordersDs.commit();
   }
}
```



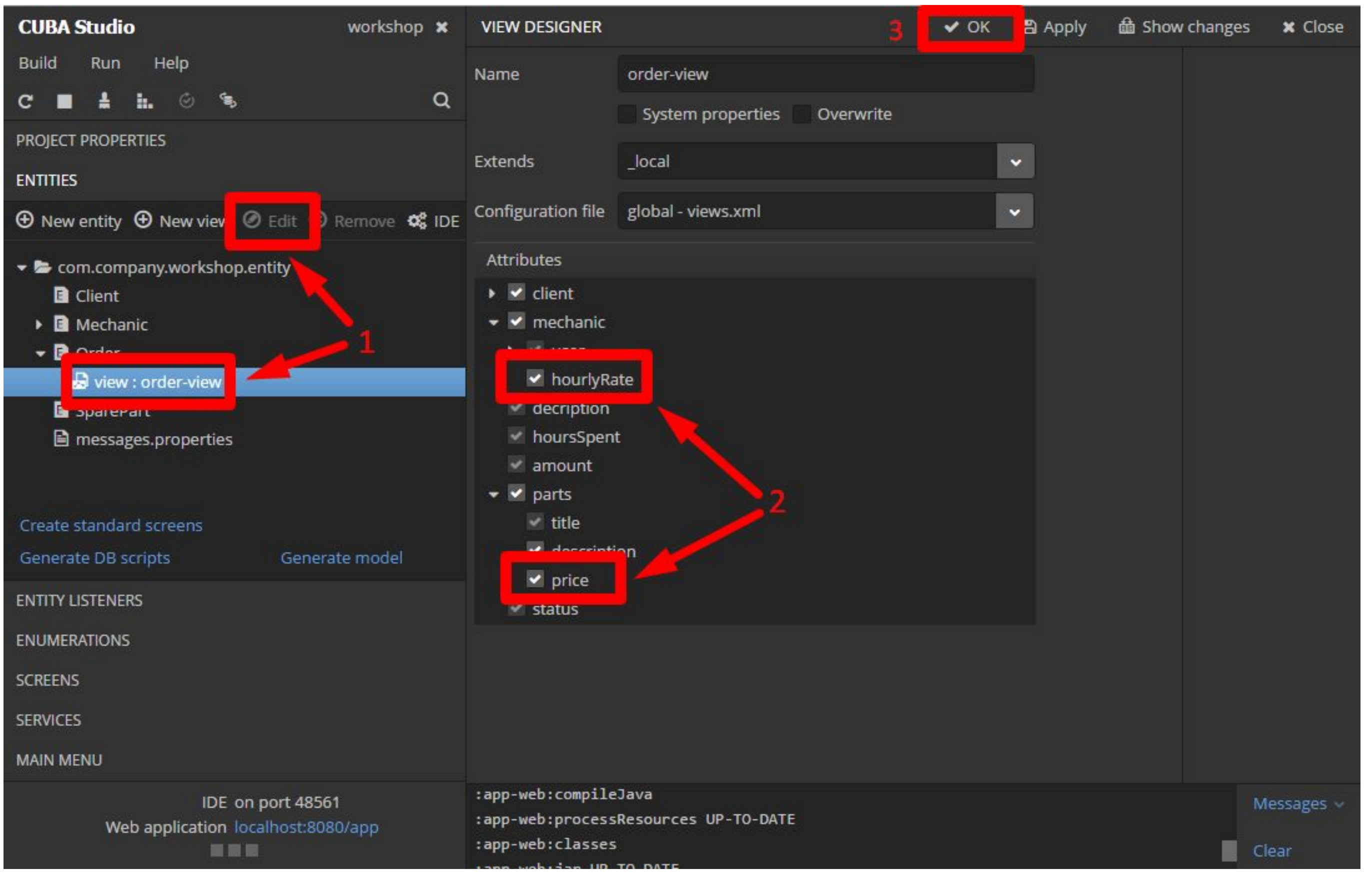
```java
package com.company.workshop.web.order;

import com.company.workshop.entity.Order;
import com.company.workshop.entity.OrderStatus;
import com.haulmont.cuba.gui.components.AbstractLookup;
import com.haulmont.cuba.gui.components.Component;
import com.haulmont.cuba.gui.data.CollectionDatasource;

import javax.inject.Inject;
import java.util.UUID;

public class OrderBrowse extends AbstractLookup {


    @Inject
    private CollectionDatasource<Order, UUID> ordersDs;

    public void onBtnNewStatusClick(Component source) {
        Order selectedItem = ordersDs.getItem();
        if (selectedItem != null) {
            selectedItem.setStatus(OrderStatus.NEW);
            ordersDs.commit();
        }
    }
```

8. In the same way create two more buttons: **Set as In Progress** and **Set as Ready**
9. **Open/Reopen** the Orders screen to see the mystery of hotdeploy. Now our mechanics can work with maximum productivity

6.3 Business Logic Development

As the next step, let's add business logic to our system to calculate the order price when we save it in the edit screen. The amount will be based on the spare parts price and time spent by the mechanic.

1. To use mechanic hourly rate and prices for parts, we'll need to load this attribute, so we need to add it to **order-view**. In order to change the view switch to the Studio, open the Entities section of the Studio navigation panel, select the **order-view** item and click **Edit**
2. Include the **hourlyRate** and **price** for parts attributes to the view
3. Click **OK** to save the view. From now we will have access to the *hourlyRate* and *price* attributes from the orders screens (editor and browser)



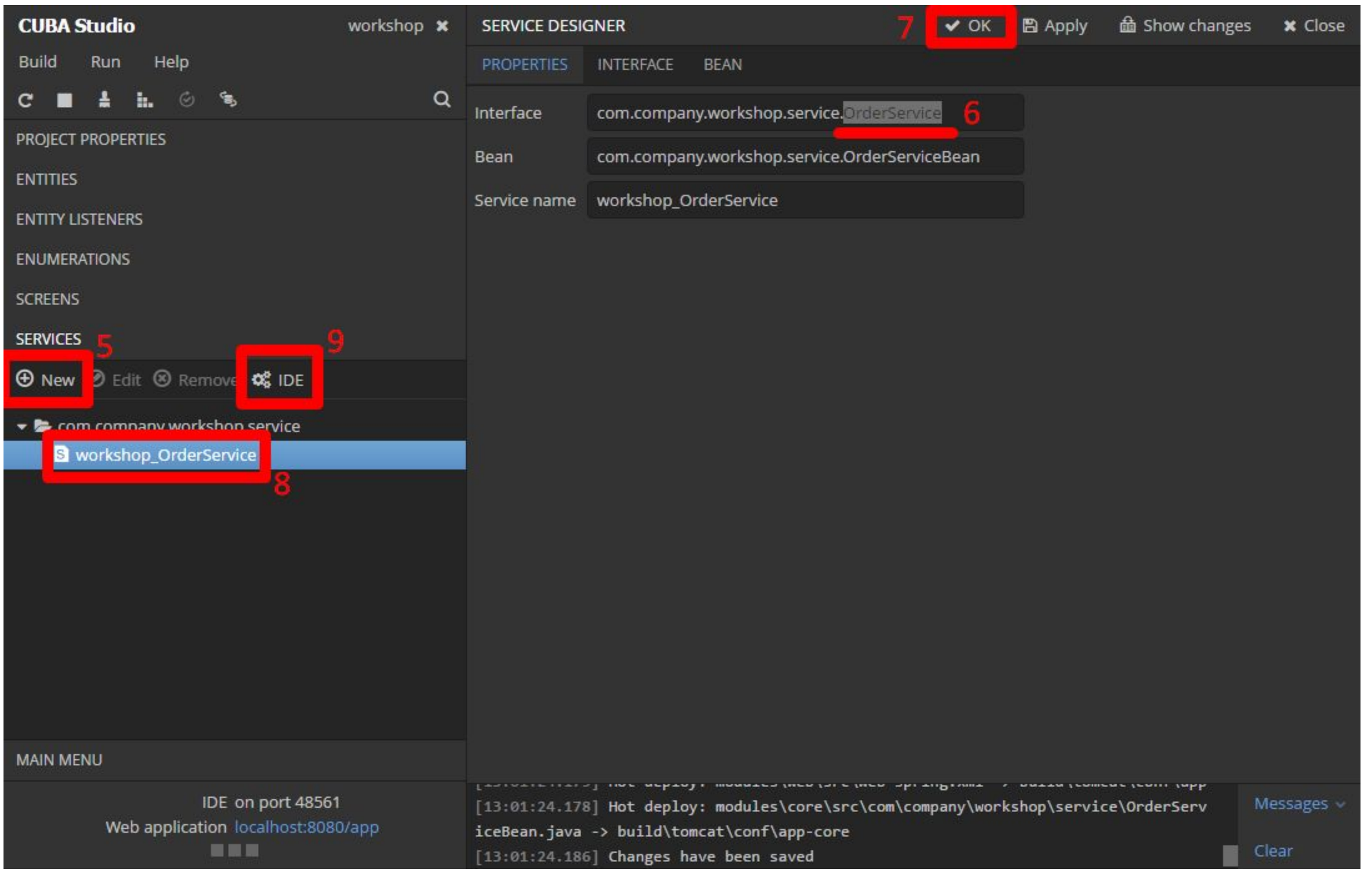
Name: order-view

Extends: _local

Configuration file: global - views.xml

```
:app-web:compileJava
:app-web:processResources UP-TO-DATE
:app-web:classes
:app-web:jar UP-TO-DATE
```

4. Go to the **Services** section in the Studio
5. Click **New**
6. Change the last part of Interface name to ***OrderService***
7. Click **OK**. The interface will be located in the global module, its implementation - in the core module. The service will be available for invocation for all clients that are connected to the middle tier of our application (web-client, portal, mobile clients or integration with third-party applications)
8. Select the **OrderService** item in the navigation panel
9. Click **IDE**



Interface: com.company.workshop.service.OrderService

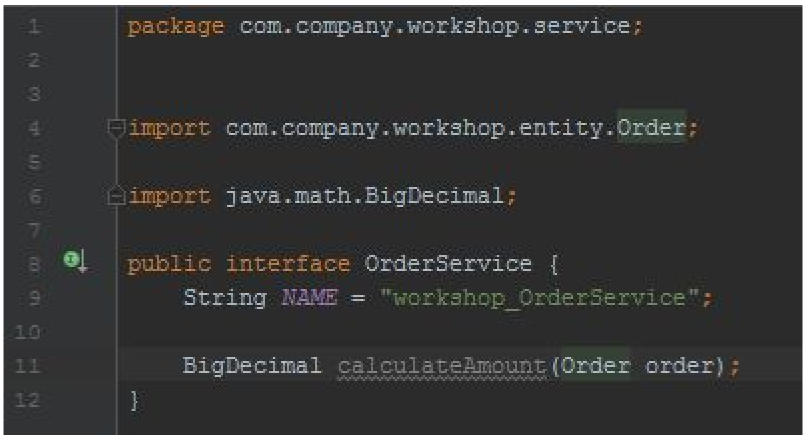
Bean: com.company.workshop.service.OrderServiceBean

Service name: workshop_OrderService

IDE on port 48561
Web application localhost:8080/app

```
[13:01:24.178] Hot deploy: modules\core\src\com\company\workshop\service\OrderServ
iceBean.java -> build\tomcat\conf\app-core
[13:01:24.186] Changes have been saved
```

10. In the Intellij IDEA, we'll see the service interface, let's add the amount calculation method to it: ***BigDecimal calculateAmount(Order order)***

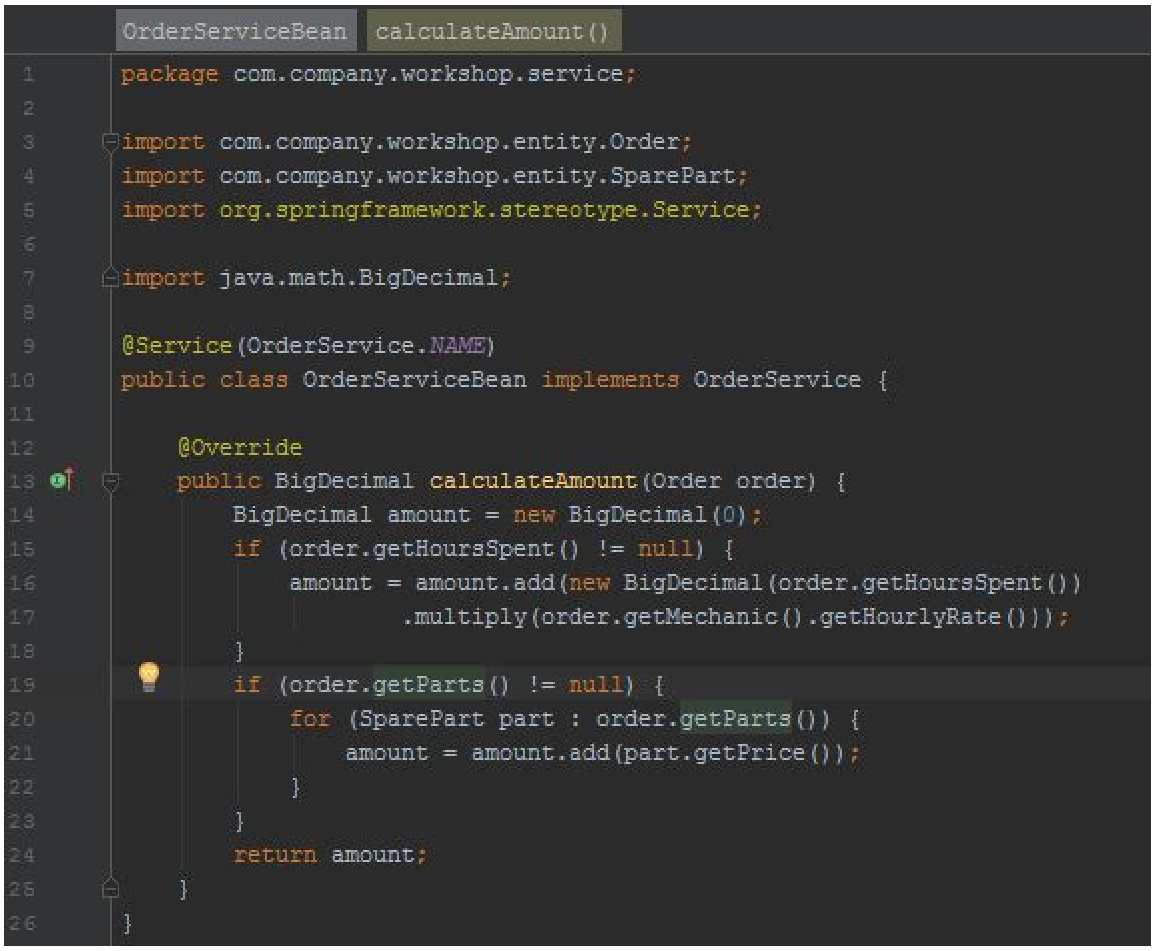


```
package com.company.workshop.service;


import com.company.workshop.entity.Order;

import java.math.BigDecimal;

public interface OrderService {
    String NAME = "workshop_OrderService";

    BigDecimal calculateAmount(Order order);
}
```

11. Go to **OrderServiceBean** using the green navigation icon at the left
12. Implement the *caclulateAmount* method

```
@Service(OrderService.NAME)
public class OrderServiceBean implements OrderService {

   @Override
   public BigDecimal calculateAmount(Order order) {
       BigDecimal amount = new BigDecimal(0);
       if (order.getHoursSpent() != null) {
           amount = amount.add(new BigDecimal(order.getHoursSpent())
                   .multiply(order.getMechanic().getHourlyRate()));
       }
```

```
            if (order.getParts() != null) {
                for (SparePart part : order.getParts()) {
                    amount = amount.add(part.getPrice());
                }
            }
            return amount;
        }
    }
```



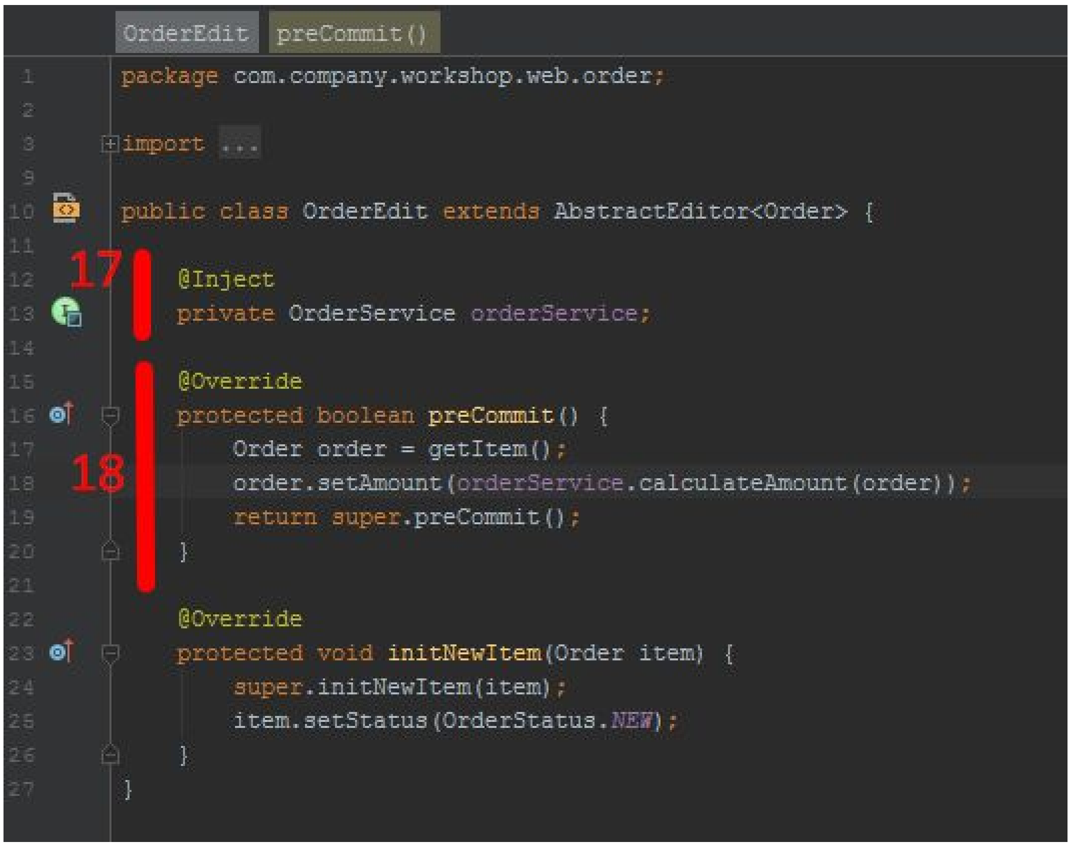
```
OrderServiceBean    calculateAmount()

package com.company.workshop.service;

import com.company.workshop.entity.Order;
import com.company.workshop.entity.SparePart;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;

@Service(OrderService.NAME)
public class OrderServiceBean implements OrderService {

    @Override
    public BigDecimal calculateAmount(Order order) {
        BigDecimal amount = new BigDecimal(0);
        if (order.getHoursSpent() != null) {
            amount = amount.add(new BigDecimal(order.getHoursSpent())
                    .multiply(order.getMechanic().getHourlyRate()));
        }
        if (order.getParts() != null) {
            for (SparePart part : order.getParts()) {
                amount = amount.add(part.getPrice());
            }
        }
        return amount;
    }
}
```

13. Go back to the Studio
14. Select the **order-edit.xml** screen in the **Screens** section of the navigation panel
15. Click **IDE**
16. Go to the screen controller (**OrderEdit** class)
17. Add **OrderService** field to class and annotate it with @Inject annotation

```
@Inject
private OrderService orderService;
```

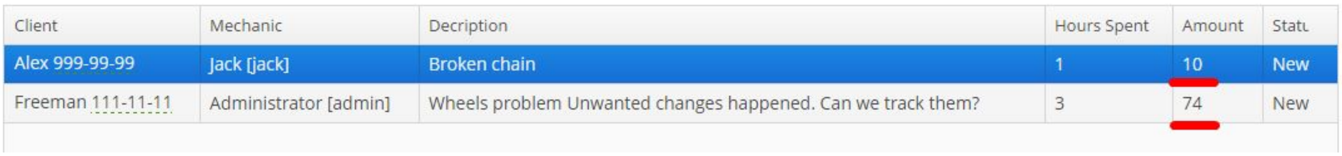
18. Override the ***preCommit()*** method and invoke the calculation method of ***OrderService***

```
@Override
protected boolean preCommit() {
    Order order = getItem();
    order.setAmount(orderService.calculateAmount(order));
    return super.preCommit();
}
```



```java
package com.company.workshop.web.order;

import ...

public class OrderEdit extends AbstractEditor<Order> {

    @Inject
    private OrderService orderService;

    @Override
    protected boolean preCommit() {
        Order order = getItem();
        order.setAmount(orderService.calculateAmount(order));
        return super.preCommit();
    }

    @Override
    protected void initNewItem(Order item) {
        super.initNewItem(item);
        item.setStatus(OrderStatus.NEW);
    }
}
```

19. Restart your application using the **Run — Restart application** action from the Studio
20. Open **Application — Orders** from the menu
21. Open **editor screen** for any order
22. Set **Hours Spent**
23. Click **OK** to save order
24. We can see a newly calculated value of the amount in the table



Client	Mechanic	Decription	Hours Spent	Amount	Statu
Alex 999-99-99	Jack [jack]	Broken chain	1	10	New
Freeman 111-11-11	Administrator [admin]	Wheels problem Unwanted changes happened. Can we track them?	3	74	New

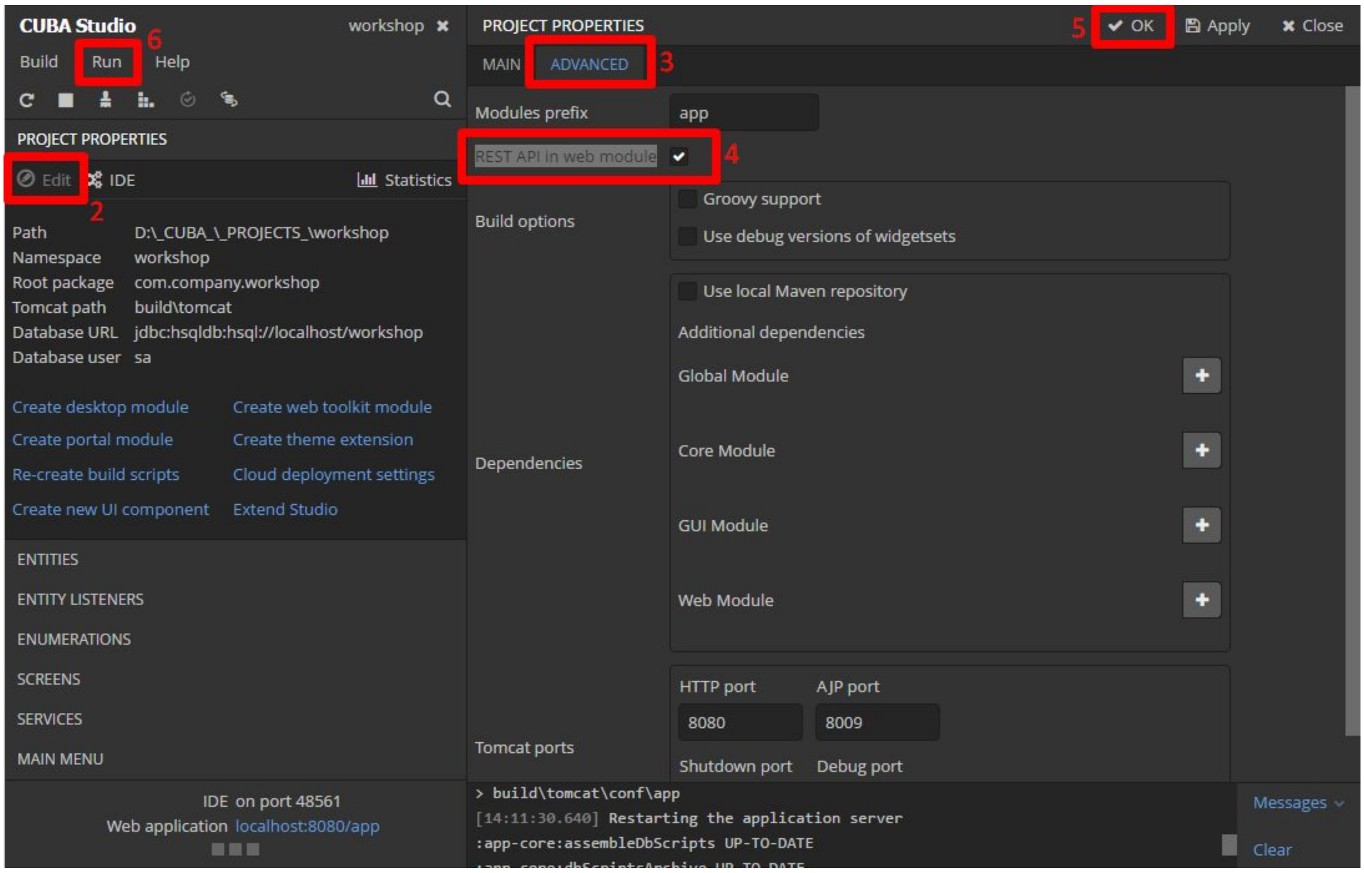
Now we can automatically calculate price based on spare parts used and time elapsed as it was mentioned in our functional specification!

6.4 REST API

Often enterprise systems have more than one client. In our example we have web client for our back office staff. In the future we could come with an idea of having a portal and/or a mobile application for our customers. For this purpose CUBA provides developers with the generic REST API.

1. Let's enable the Generic REST API in our application. Go to the **Project Properties** section of the navigation panel in the CUBA Studio
2. Press **Edit**
3. Move to the **Advanced** tab
4. Select the REST API in web module checkbox
5. Click **OK**

6. Restart the application using the **Run — Restart application** action from the main menu of the Studio



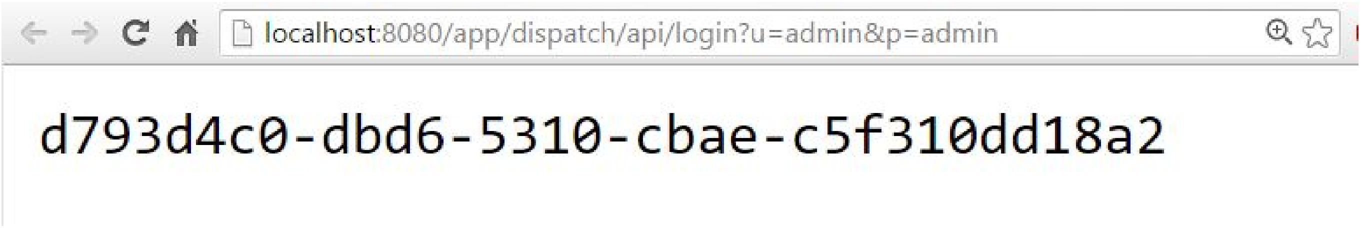
Path D:_CUBA__PROJECTS_\workshop
Namespace workshop
Root package com.company.workshop
Tomcat path build\tomcat
Database URL jdbc:hsqldb:hsql://localhost/workshop
Database user sa

IDE on port 48561
Web application localhost:8080/app

Modules prefix: app
REST API in web module: ✓
Build options: Groovy support; Use debug versions of widgetsets
Dependencies: Use local Maven repository; Additional dependencies: Global Module, Core Module, GUI Module, Web Module
Tomcat ports: HTTP port 8080, AJP port 8009, Shutdown port, Debug port

```
> build\tomcat\conf\app
[14:11:30.640] Restarting the application server
:app-core:assembleDbScripts UP-TO-DATE
```

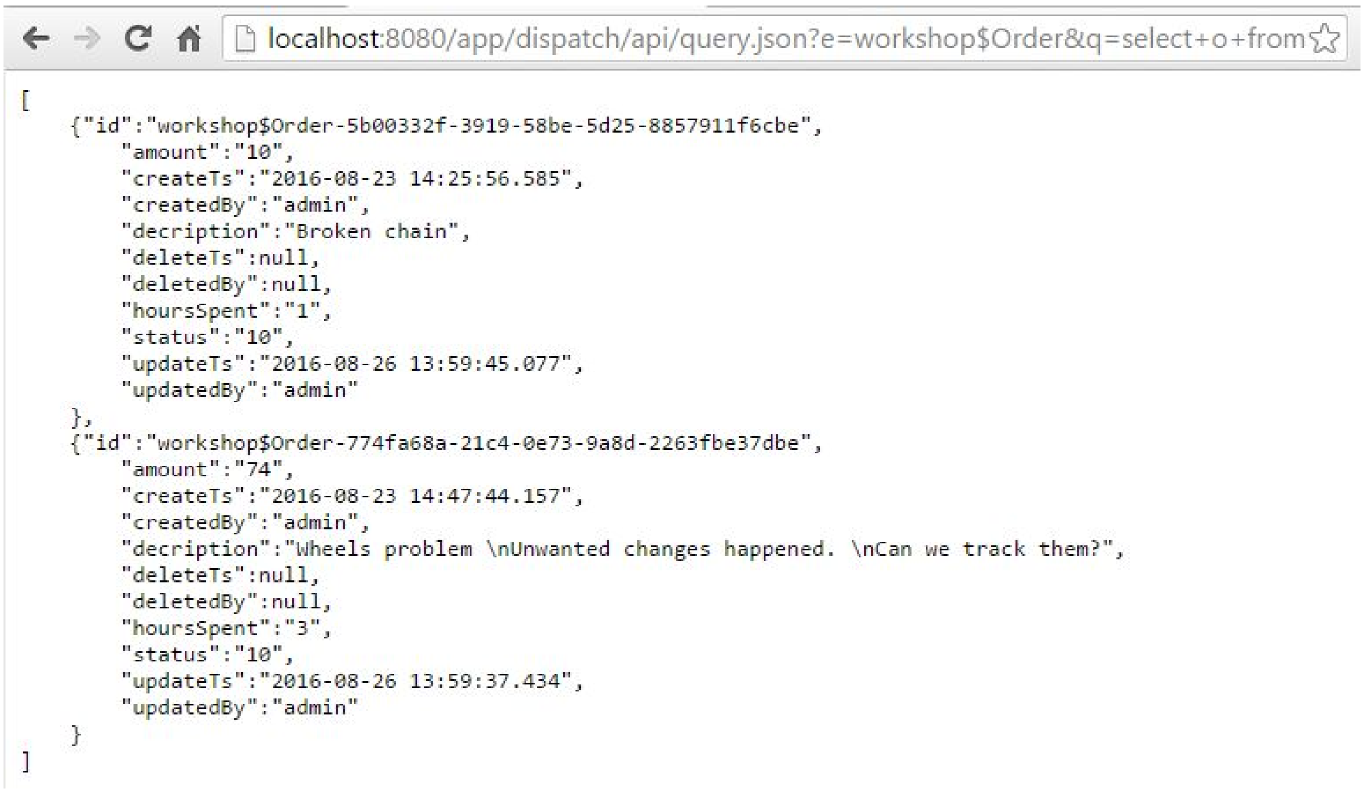
7. Let's try to get a list of orders using REST-API.
   To start working with REST-API, you need to get the middle layer session using the login method as CUBA don't allow unauthorized access to data. You can invoke the login method right from the browser address bar: http://localhost:8080/app/dispatch/api/login?u=admin&p=admin.
   Save the returned value, that represents your session id, because we will need it for the following request



```
localhost:8080/app/dispatch/api/login?u=admin&p=admin
```

```
d793d4c0-dbd6-5310-cbae-c5f310dd18a2
```

8. Let's load the list of new orders in JSON using the following query:
   select o from workshop$Order o where o.status = 10
   REST-API request for this query will look as:
   http://localhost:8080/app/dispatch/api/query.json?e=workshop$Order&q=select+o+from+workshop$Order+o+where+o.status=10&s=[INSERT YOUR SESSION ID FROM THE PREVIOUS STEP]


localhost:8080/app/dispatch/api/query.json?e=workshop$Order&q=select+o+from

```
[
    {"id":"workshop$Order-5b00332f-3919-58be-5d25-8857911f6cbe",
        "amount":"10",
        "createTs":"2016-08-23 14:25:56.585",
        "createdBy":"admin",
        "decription":"Broken chain",
        "deleteTs":null,
        "deletedBy":null,
        "hoursSpent":"1",
        "status":"10",
        "updateTs":"2016-08-26 13:59:45.077",
        "updatedBy":"admin"
    },
    {"id":"workshop$Order-774fa68a-21c4-0e73-9a8d-2263fbe37dbe",
        "amount":"74",
        "createTs":"2016-08-23 14:47:44.157",
        "createdBy":"admin",
        "decription":"Wheels problem \nUnwanted changes happened. \nCan we track them?",
        "deleteTs":null,
        "deletedBy":null,
        "hoursSpent":"3",
        "status":"10",
        "updateTs":"2016-08-26 13:59:37.434",
        "updatedBy":"admin"
    }
]
```

That was the last requirement from the functional specification! Now the system is ready for production!

7 CONCLUSION

This is very small application for bicycle workshop management. It is simple, but can be applied for a real local workshop.

You can run it in production environment (including clouds) as is and it will be suitable for its purpose.

You can add much more functionality using CUBA additional modules, and this enables you to grow your application to a big strong solution.

WYSI WYG
High Availability
Data Access Control
Reporting
Charts
LDAP Integration
Generic REST API
Scheduled Tasks
Audit
Generic Data Filter
BPM
Java IDE
aPaaS Deployment
Admin Tools
Visual Components
Google Maps
Credit Cards
Scaffolding
Hot Deploy
User Management
Dynamic Attributes
Multi-Language
Full-text Search