CUBA.platform
www.cuba-platform.com

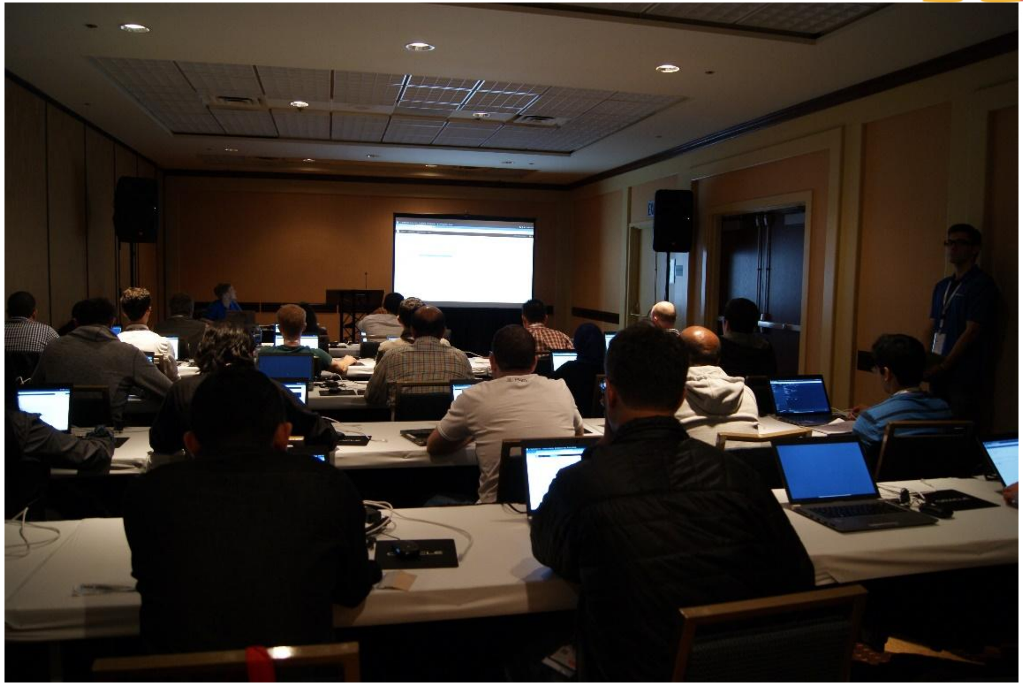
Develop a Fully Functional Business Application in Hours with CUBA Platform

Objectives

This document will guide you through the key features of the CUBA Platform framework and show how you can accelerate development of enterprise applications in the format of Hands-on-Labs.

Estimated time to complete this lab is 3 hours.

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

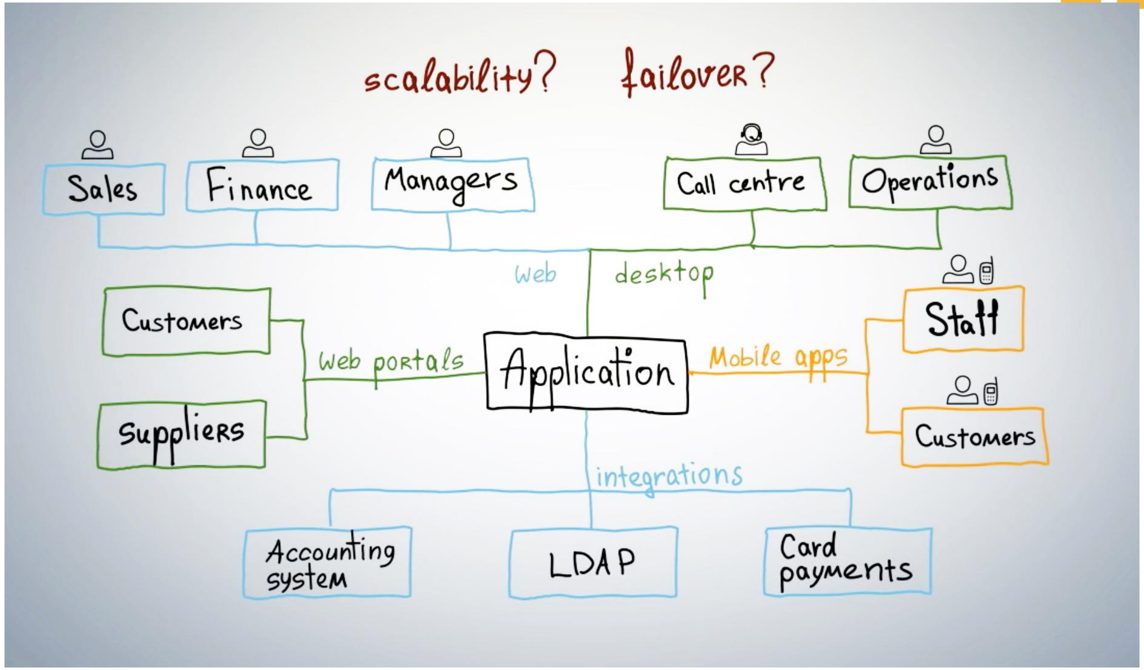


CUBA.platform
www.cuba-platform.com

What is CUBA Platform?

A high level Java framework for rapid enterprise software development. The platform provides a rich set of features:

- Rich web/desktop UI
- CRUD
- Role based and row level security
- Reporting
- Charts
- Full text search
- REST-API
- Scalable deployment



www.cuba-platform.com

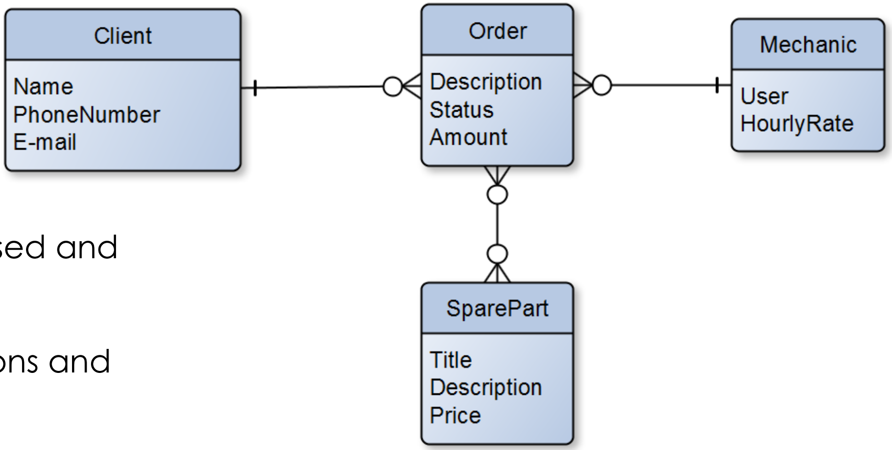
What we are going to automate

Application for a small bicycle workshop

Short functional specification from the application:

- Store customers with their name, mobile phone and email
- Customer email to be used to notify about order status
- Record information about orders: price for repair and time spent by mechanic
- Keep track of spare parts in stock and enable search for parts
- Automatically calculate price based on spare parts used and time elapsed
- Control security permissions for screens, CRUD operations and records' attributes
- Audit of critical data changes
- Charts and reports

The data model



Application features

Our application will:

- Have Rich Web UI, with Ajax communication
- Perform basic CRUD operations
- Contain the business logic for calculating prices
- Manage user access rights
- Present data in the form of reports and charts
- Have audit capabilities
- Allow us to create mobile applications or website using REST-API

**Just two hours -
and we are ready for production!**

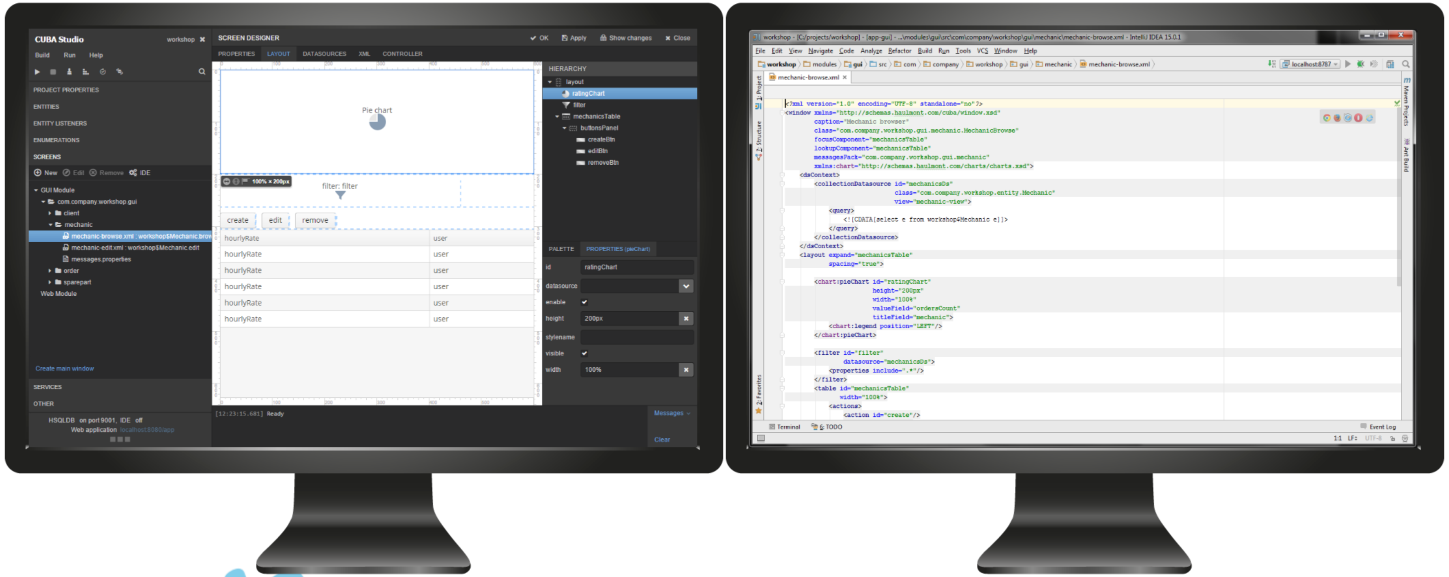
www.cuba-platform.com

Environment and tools

CUBA.platform
www.cuba-platform.com

Development environment

1. Download CUBA Studio https://www.cuba-platform.com/download
2. Install IntelliJ IDEA
3. Install CUBA Plugin for IntelliJ IDEA

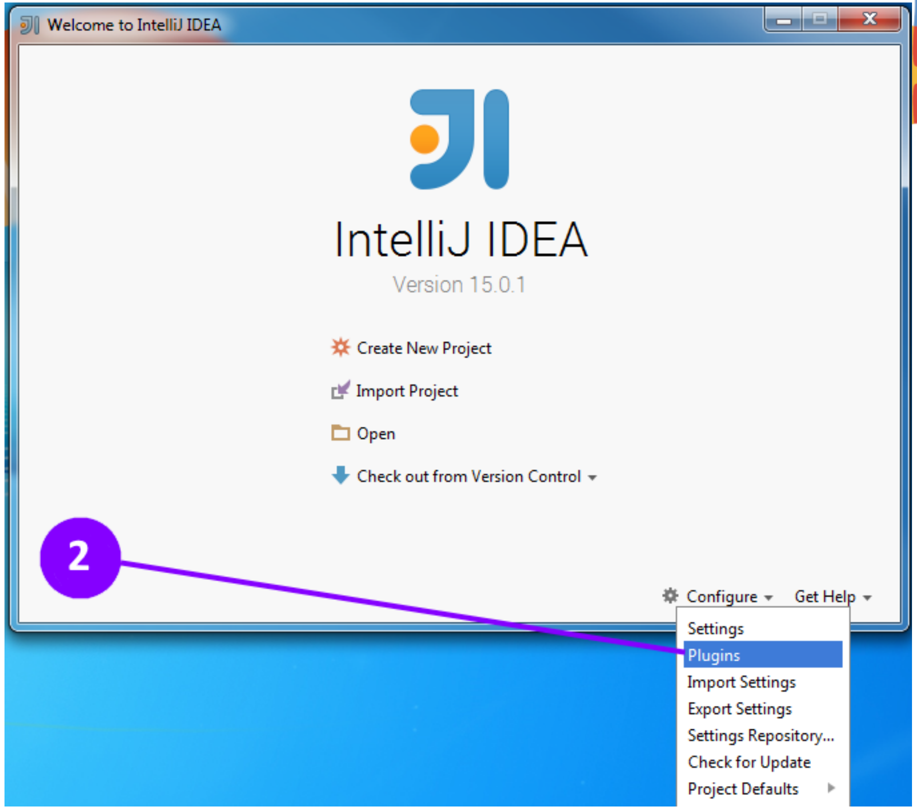


```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<window xmlns="http://schemas.haulmont.com/cuba/window.xsd"
        caption="Mechanic browser"
        class="com.company.workshop.gui.mechanic.MechanicBrowse"
        focusComponent="mechanicsTable"
        lookupComponent="mechanicsTable"
        messagesPack="com.company.workshop.gui.mechanic"
        xmlns:chart="http://schemas.haulmont.com/charts/charts.xsd">
    <dsContext>
        <collectionDatasource id="mechanicsDs"
                              class="com.company.workshop.entity.Mechanic"
                              view="mechanic-view">
            <query>
                <![CDATA[select e from workshop$Mechanic e]]>
            </query>
        </collectionDatasource>
    </dsContext>
    <layout expand="mechanicsTable"
            spacing="true">

        <chart:pieChart id="ratingChart"
                        height="200px"
                        width="100%"
                        valueField="ordersCount"
                        titleField="mechanic">
            <chart:legend position="LEFT"/>
        </chart:pieChart>

        <filter id="filter"
                datasource="mechanicsDs">
            <properties include=".*"/>
        </filter>
        <table id="mechanicsTable"
               width="100%">
            <actions>
                <action id="create"/>
```

CUBA.platform
www.cuba-platform.com

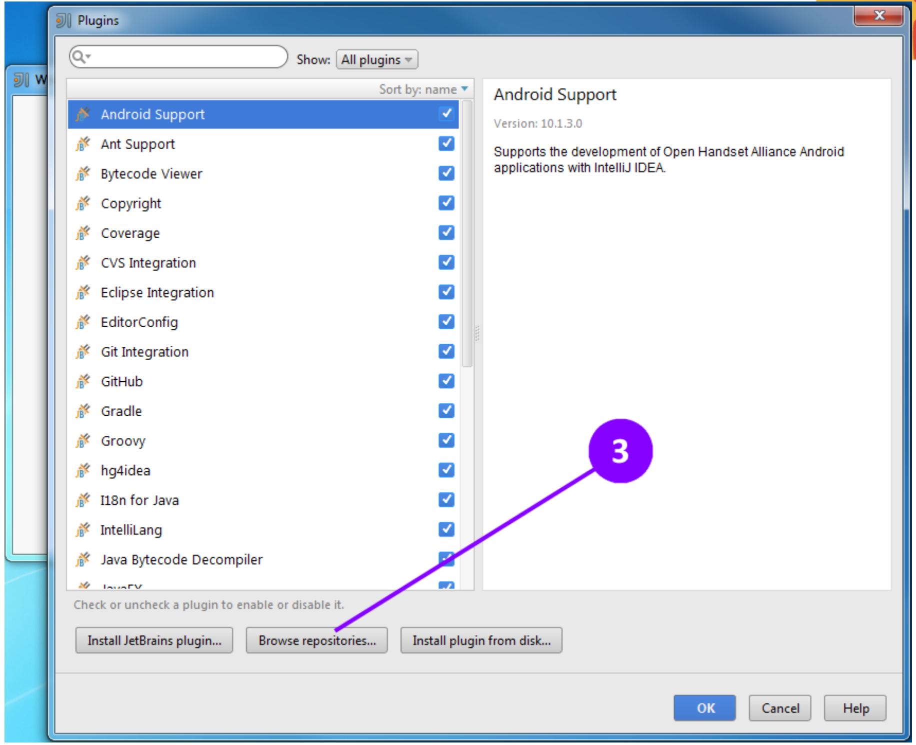
How to install CUBA Plugin for IntelliJ IDEA

1. Run IntelliJ IDEA
2. Open menu Configure - Plugins



How to install CUBA Plugin for IntelliJ IDEA

3. Click on Browse repositories



Android Support ✓
Ant Support ✓
Bytecode Viewer ✓
Copyright ✓
Coverage ✓
CVS Integration ✓
Eclipse Integration ✓
EditorConfig ✓
Git Integration ✓
GitHub ✓
Gradle ✓
Groovy ✓
hg4idea ✓
I18n for Java ✓
IntelliLang ✓
Java Bytecode Decompiler ✓

Android Support

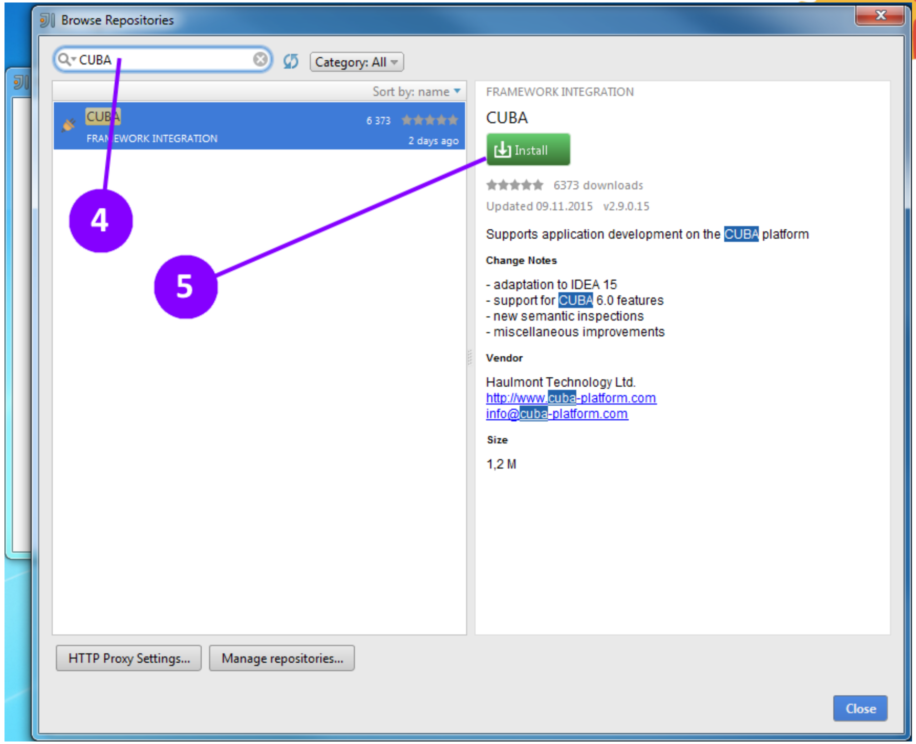
Version: 10.1.3.0

Supports the development of Open Handset Alliance Android applications with IntelliJ IDEA.

Check or uncheck a plugin to enable or disable it.

How to install CUBA Plugin for IntelliJ IDEA

4. Find CUBA plugin
5. Click Install



FRAMEWORK INTEGRATION

CUBA

Install

6373 downloads

Updated 09.11.2015 v2.9.0.15

Supports application development on the CUBA platform

Change Notes

- adaptation to IDEA 15
- support for CUBA 6.0 features
- new semantic inspections
- miscellaneous improvements

Vendor

Haulmont Technology Ltd.
http://www.cuba-platform.com
info@cuba-platform.com

Size

1,2 M

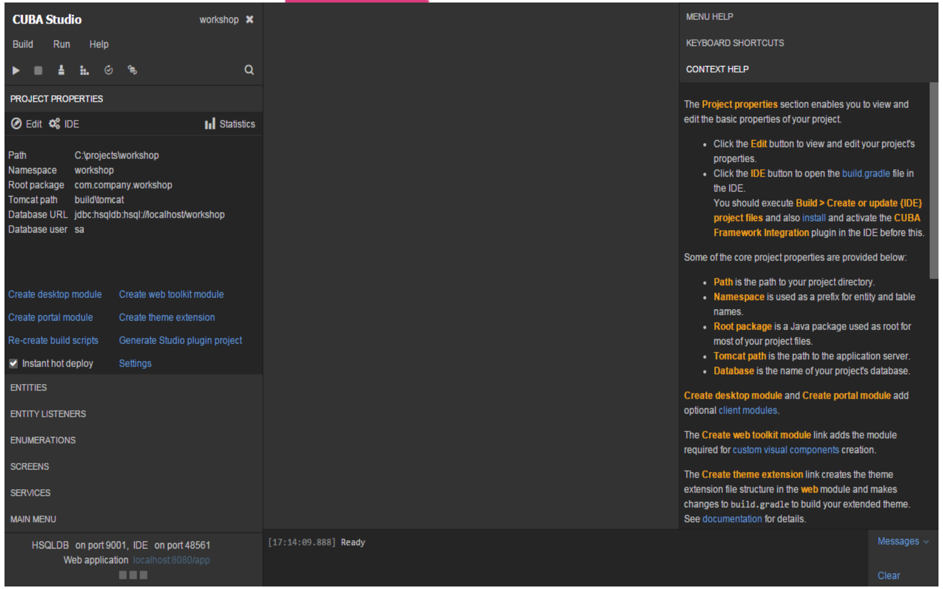
CUBA.platform
www.cuba-platform.com

Getting started

What is CUBA Studio?

CUBA Studio – a web based development tool that

- Offers a quick way to configure a project and describe data model
- Manages DB scripts
- Enables scaffolding and visual design for the user interface
- Works in parallel with your favorite IDE:
  IntelliJ IDEA or Eclipse



CUBA Studio workshop

Build Run Help

PROJECT PROPERTIES

Edit IDE Statistics

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

Create desktop module · Create web toolkit module
Create portal module · Create theme extension
Re-create build scripts · Generate Studio plugin project
Instant hot deploy · Settings

ENTITIES
ENTITY LISTENERS
ENUMERATIONS
SCREENS
SERVICES
MAIN MENU

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

```
[17:14:09.888] Ready
```

MENU HELP
KEYBOARD SHORTCUTS
CONTEXT HELP

The **Project properties** section enables you to view and edit the basic properties of your project.

- Click the **Edit** button to view and edit your project's properties.
- Click the **IDE** button to open the build.gradle file in the IDE.
  You should execute **Build > Create or update {IDE} project files** and also install and activate the **CUBA Framework Integration** plugin in the IDE before this.

Some of the core project properties are provided below:

- **Path** is the path to your project directory.
- **Namespace** is used as a prefix for entity and table names.
- **Root package** is a Java package used as root for most of your project files.
- **Tomcat path** is the path to the application server.
- **Database** is the name of your project's database.

Create desktop module and **Create portal module** add optional client modules.

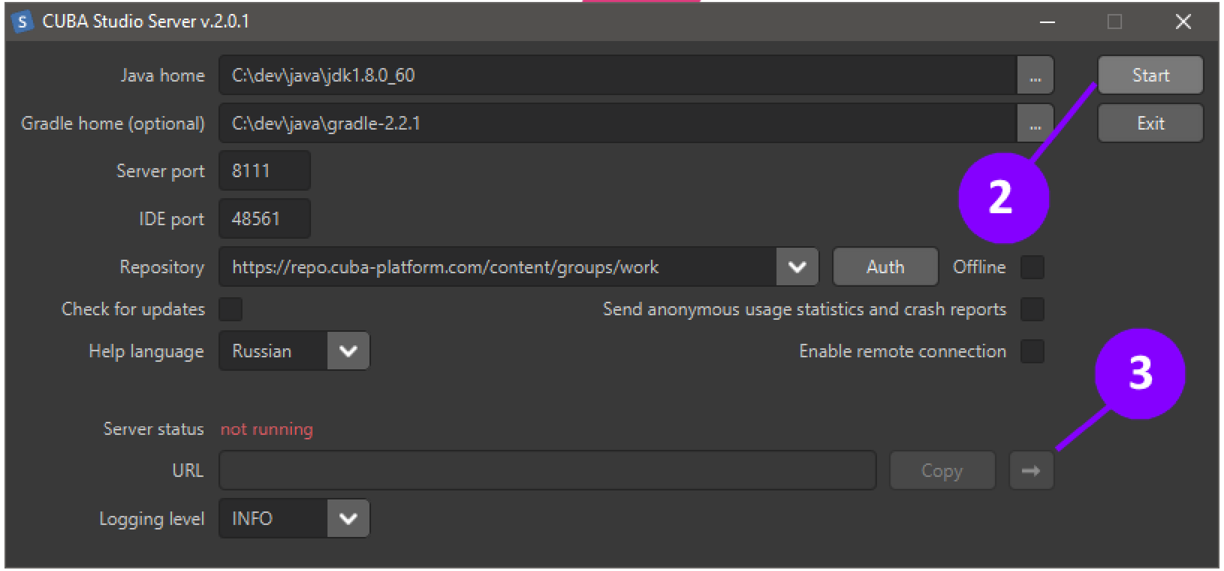
The **Create web toolkit module** link adds the module required for custom visual components creation.

The **Create theme extension** link creates the theme extension file structure in the **web** module and makes changes to `build.gradle` to build your extended theme. See documentation for details.

Messages
Clear

Start CUBA Studio

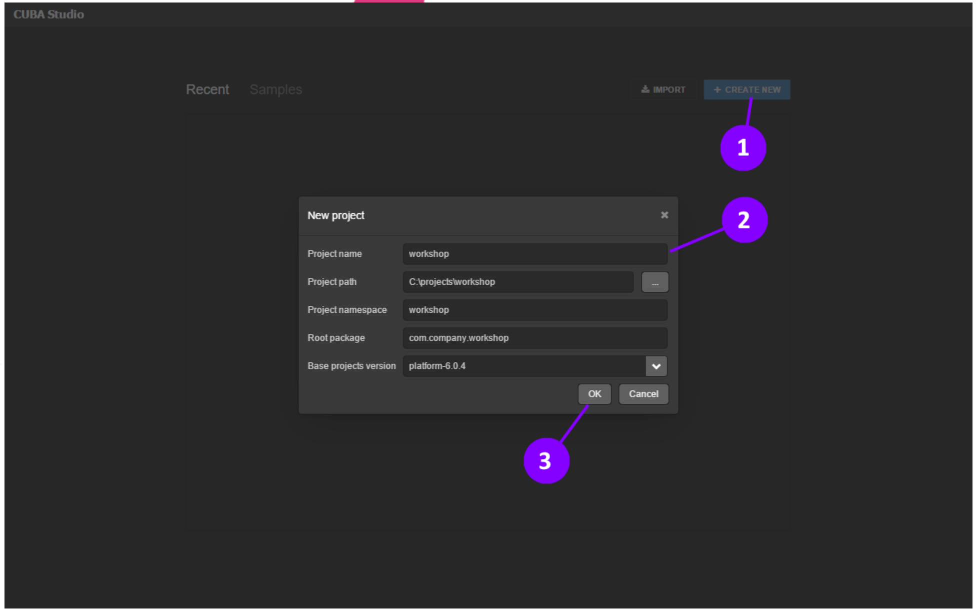
1. Run CUBA Studio
2. Click **Start** in the launcher window
3. Go to the browser by clicking the Arrow button


CUBA Studio Server v.2.0.1

Java home	C:\dev\java\jdk1.8.0_60
Gradle home (optional)	C:\dev\java\gradle-2.2.1
Server port	8111
IDE port	48561
Repository	https://repo.cuba-platform.com/content/groups/work
Check for updates	
Help language	Russian
Server status	not running
URL	
Logging level	INFO

Create a new project

1. Click **Create New** on welcome screen
2. Fill up project name: ***workshop***
3. Click **OK** and you'll get into the CUBA Studio workspace

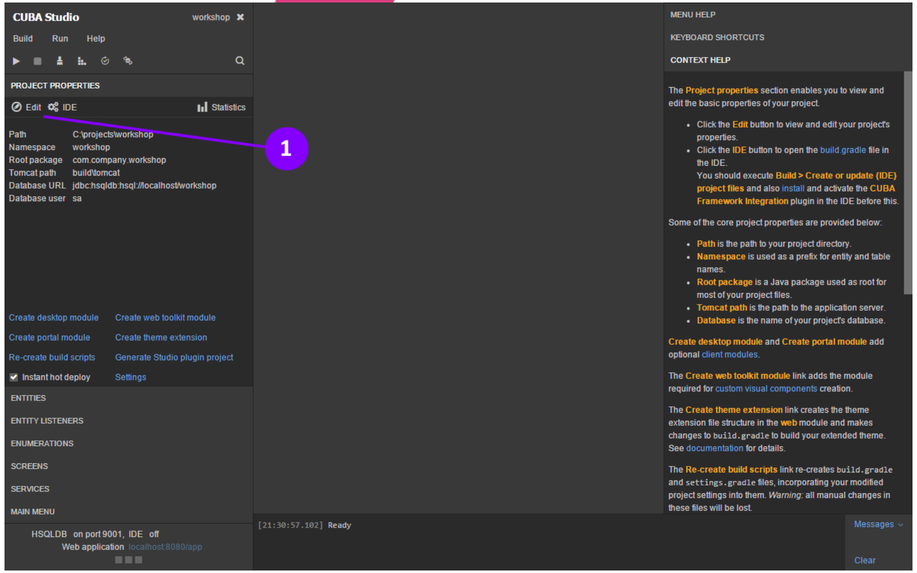


Project name: workshop
Project path: C:\projects\workshop
Project namespace: workshop
Root package: com.company.workshop
Base projects version: platform-6.0.4

CUBA Studio workspace

Using CUBA Studio you can easily create **Entities**, **Screens** and stubs for Services.
You can hide/show the **Help** panel using menu **Help - Show help panel**

1. Click **Edit** in the **Project Properties** section



Path	C:\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 off
Web application localhost:8080/app

```
[21:30:57.102] Ready
```

The Project properties section enables you to view and edit the basic properties of your project.

- Click the Edit button to view and edit your project's properties.
- Click the IDE button to open the build.gradle file in the IDE.
  You should execute Build > Create or update {IDE} project files and also install and activate the CUBA Framework Integration plugin in the IDE before this.

Some of the core project properties are provided below:

- Path is the path to your project directory.
- Namespace is used as a prefix for entity and table names.
- Root package is a Java package used as root for most of your project files.
- Tomcat path is the path to the application server.
- Database is the name of your project's database.

Create desktop module and Create portal module add optional client modules.

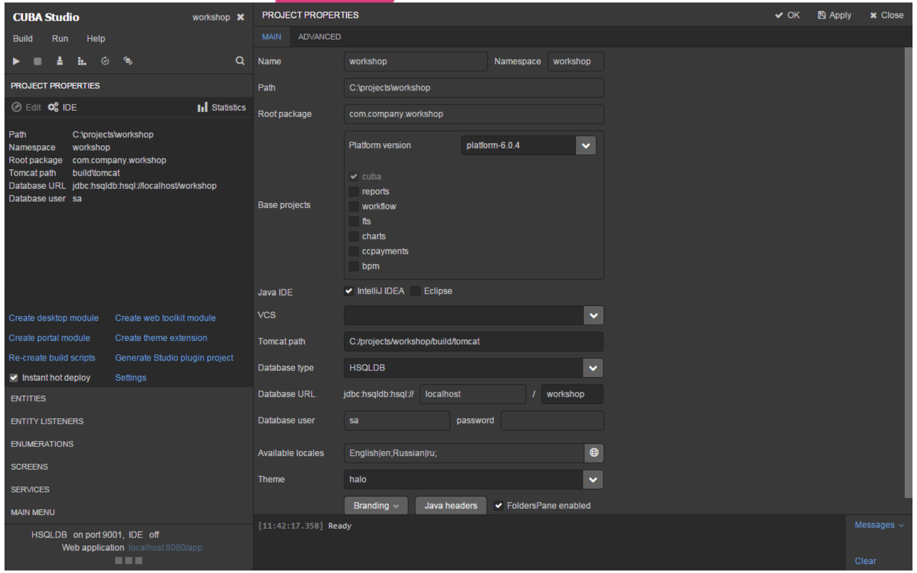
The Create web toolkit module link adds the module required for custom visual components creation.

The Create theme extension link creates the theme extension file structure in the web module and makes changes to `build.gradle` to build your extended theme. See documentation for details.

The Re-create build scripts link re-creates `build.gradle` and `settings.gradle` files, incorporating your modified project settings into them. *Warning:* all manual changes in these files will be lost.

Project properties screen

This is a page where we configure our project.
The CUBA Platform supports PostgreSQL, MS SQL, Oracle and HSQL databases.



Path	C:\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 off
Web application localhost:8080/app

PROJECT PROPERTIES

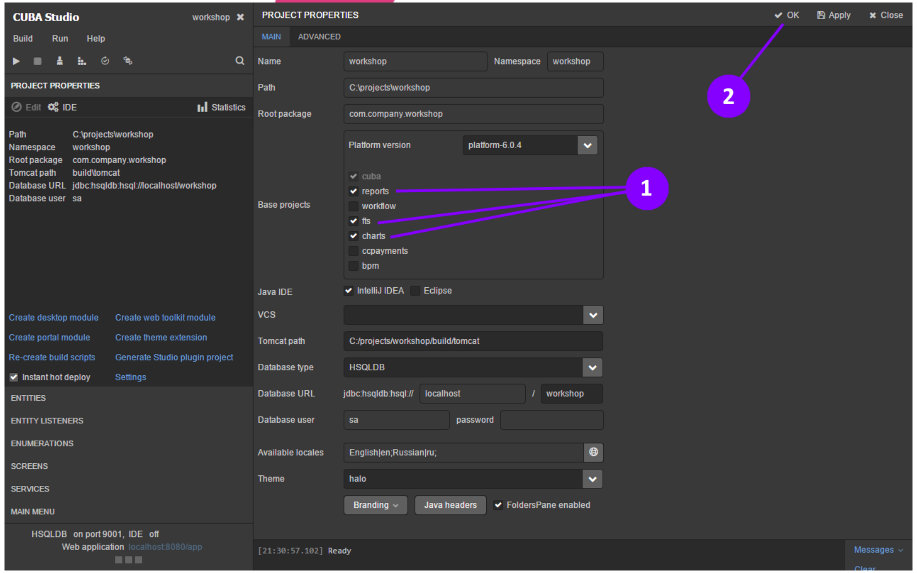
Name	workshop
Namespace	workshop
Path	C:\projects\workshop
Root package	com.company.workshop
Platform version	platform-6.0.4
Base projects	✓ cuba, reports, workflow, fts, charts, ccpayments, bpm
Java IDE	✓ IntelliJ IDEA, Eclipse
VCS	
Tomcat path	C:/projects/workshop/build/tomcat
Database type	HSQLDB
Database URL	jdbc:hsqldb:hsql:// localhost / workshop
Database user	sa
password	
Available locales	English|en;Russian|ru;
Theme	halo
FoldersPane enabled	✓

```
[11:42:17.358] Ready
```

Use required modules

1. Select checkboxes for **reports**, **fts** (full text search) and **charts** in the **Base projects** section
2. Click **OK** in the upper part of the page
3. Studio will warn us about changing the project build file, just click **OK**.

Studio will automatically add necessary dependencies and regenerate project files for IDE.



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

Create desktop module · Create web toolkit module · Create portal module · Create theme extension · Re-create build scripts · Generate Studio plugin project · ✓ Instant hot deploy · Settings

ENTITIES
ENTITY LISTENERS
ENUMERATIONS
SCREENS
SERVICES
MAIN MENU

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

Name	workshop — Namespace: workshop
Path	C:\projects\workshop
Root package	com.company.workshop
Platform version	platform-6.0.4
Base projects	✓ cuba, ✓ reports, ☐ workflow, ✓ fts, ✓ charts, ☐ ccpayments, ☐ bpm
Java IDE	✓ IntelliJ IDEA ☐ Eclipse
VCS	
Tomcat path	C:/projects/workshop/build/tomcat
Database type	HSQLDB
Database URL	jdbc:hsqldb:hsql:// localhost / workshop
Database user	sa — password
Available locales	English|en;Russian|ru;
Theme	halo
	Branding · Java headers · ✓ FoldersPane enabled

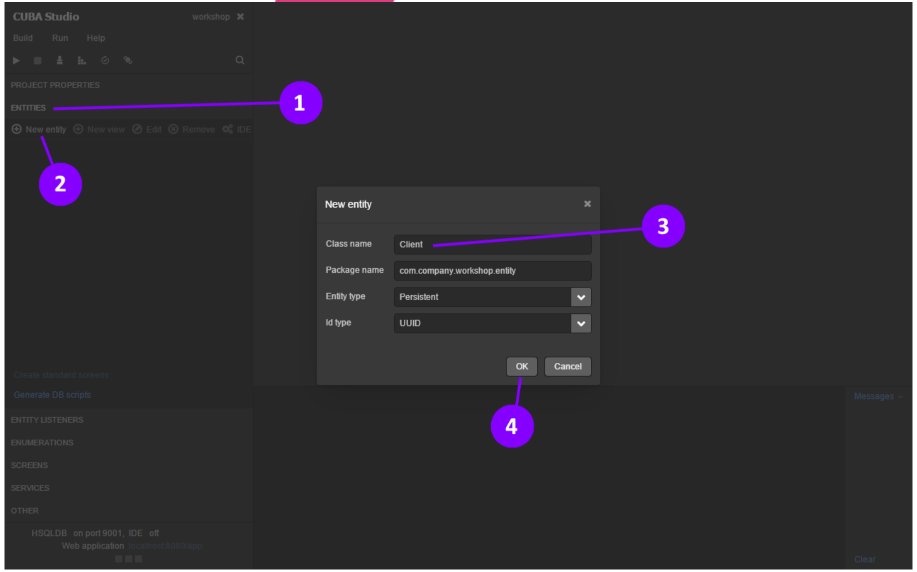
```
[21:30:57.102] Ready
```

Messages
Clear

Data model

Create the data model

1. Open the **Entities** section of the navigation panel
2. Click **New entity**
3. Input **Class name**: ***Client***
4. Click **OK**



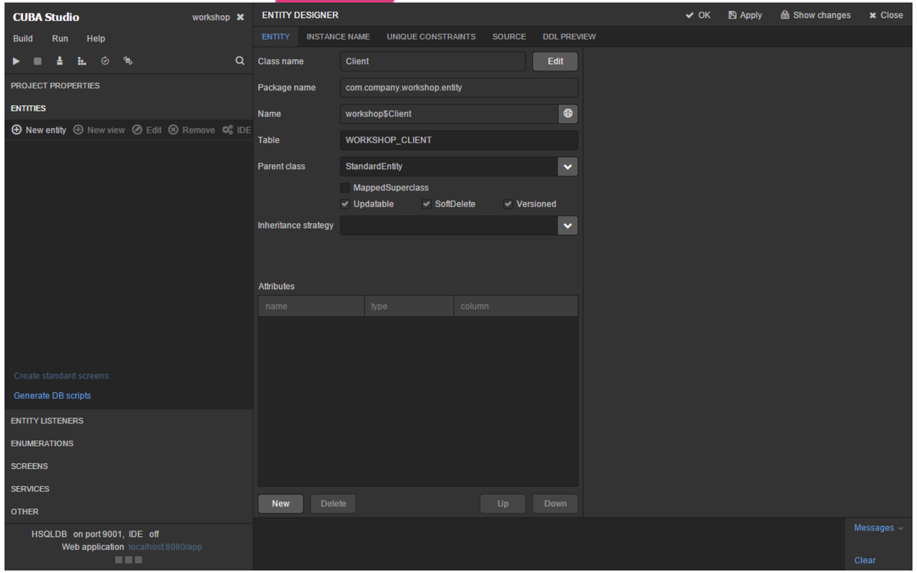
New entity

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

Entity designer

Here we can specify a parent class and corresponding table in the database, define attributes for an entity and manage other options.

Our class inherits ***StandardEntity***, the service class which supports **Soft Deletion** and contains a number of platform internal attributes (***createTs***, ***createdBy*** and others).



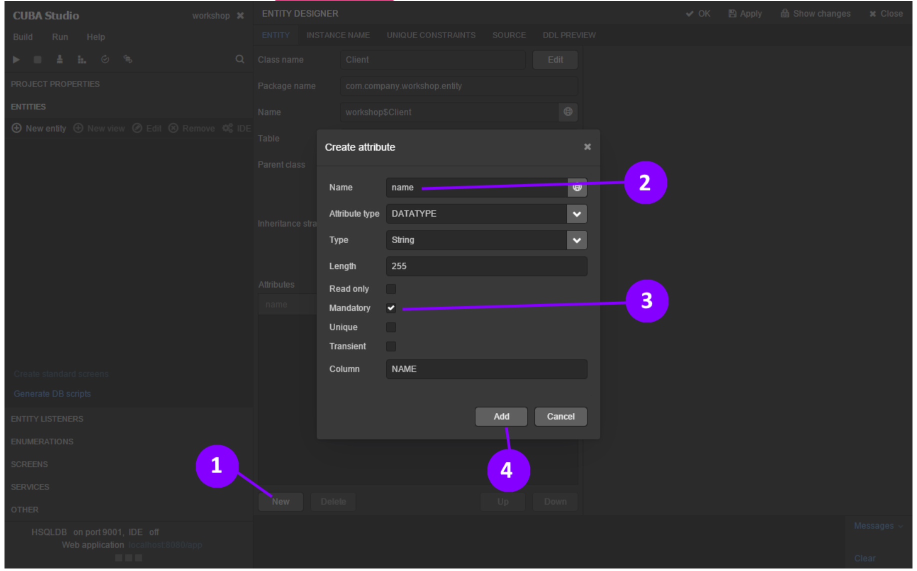
Class name: Client
Package name: com.company.workshop.entity
Name: workshop$Client
Table: WORKSHOP_CLIENT
Parent class: StandardEntity
MappedSuperclass
✓ Updatable ✓ SoftDelete ✓ Versioned
Inheritance strategy:

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

Attribute editor

1. Add a new attribute by clicking **New**
2. Enter Name: ***name***
3. Select the **Mandatory** checkbox
4. Click on **Add**

Attribute editor enables us to create or edit attribute and its parameters, such as Attribute type, Java Type, Read only, Mandatory, Unique, etc.



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

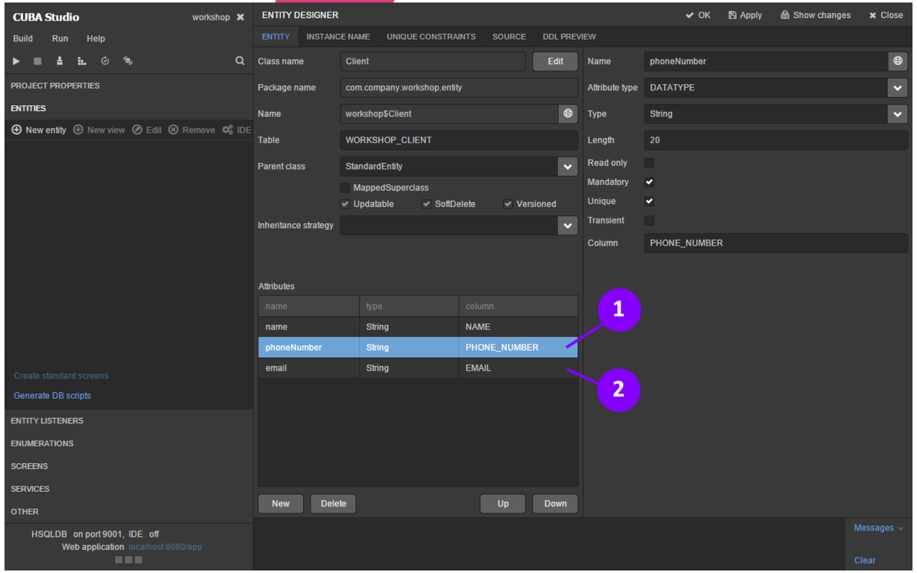
Create attribute

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

Client entity and its attributes

Similarly, we add ***phoneNumber*** and ***email***.

1. Add ***phoneNumber*** as a mandatory attribute with the length of 20 and unique flag
2. Add ***email*** as a mandatory attribute with the length of 50 and flagged as unique



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

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

Attributes

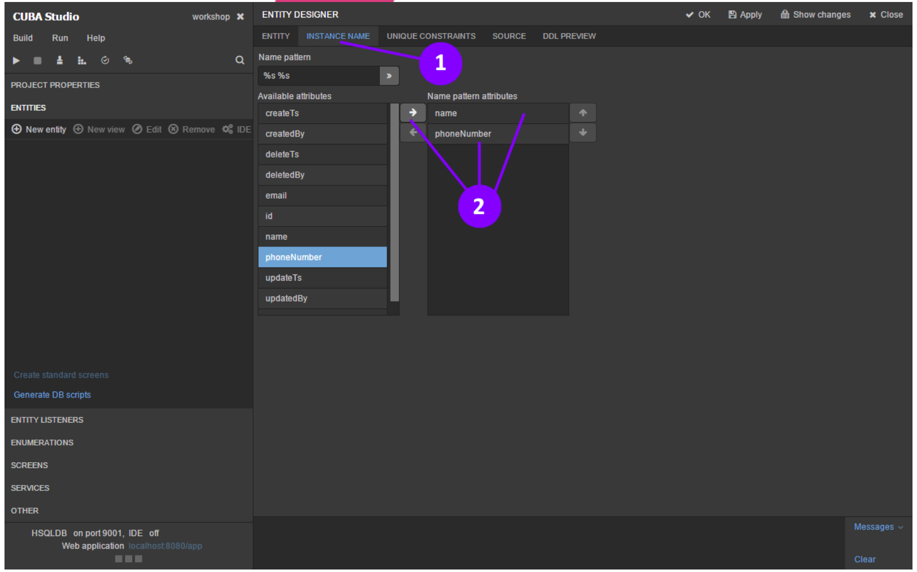
name	type	column
name	String	NAME
phoneNumber	String	PHONE_NUMBER
email	String	EMAIL

Name	phoneNumber
Attribute type	DATATYPE
Type	String
Length	20
Read only	☐
Mandatory	✓
Unique	✓
Transient	☐
Column	PHONE_NUMBER

Instance name

Instance name is a default string representation of **Entity** for user interface (tables, dropdown lists, etc).

1. Go to the **Instance name** tab
2. Select ***name*** and ***phoneNumber***

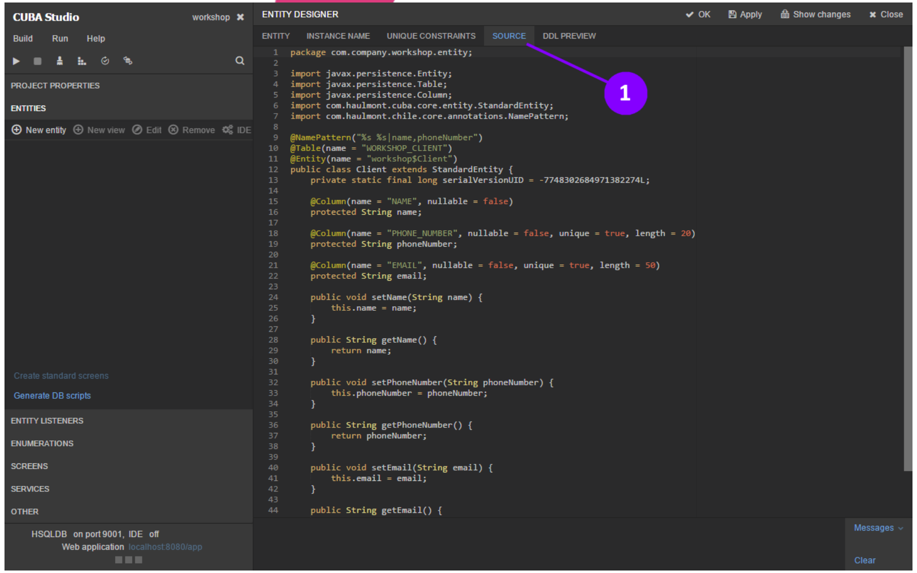


Generated source code for the Client entity

1. Click on the **Source** tab of the **Entity designer**

This is a regular Java class, annotated with the ***javax.persistence*** annotations and supplemented by CUBA annotations.

You can change source code of an entity manually and the Studio will read your changes and apply those back to model.

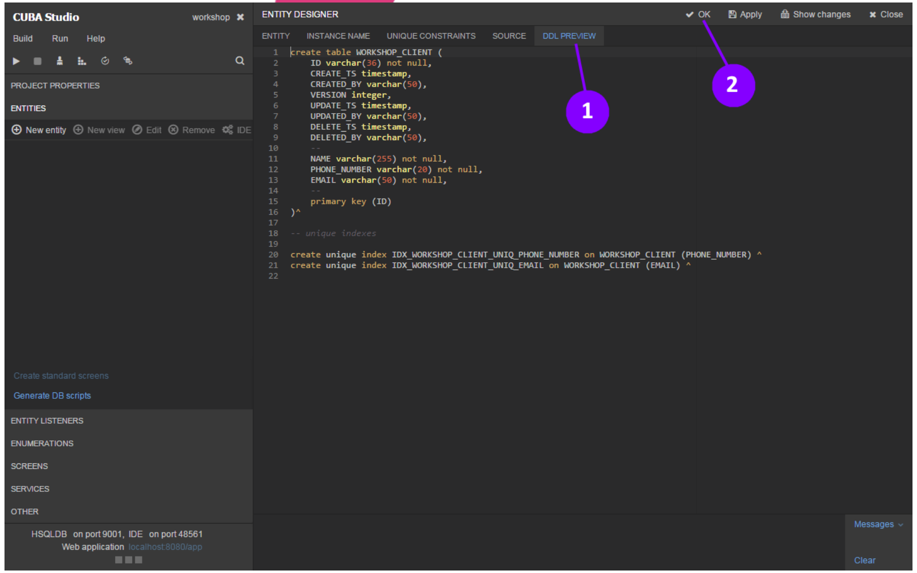


```java
package com.company.workshop.entity;

import javax.persistence.Entity;
import javax.persistence.Table;
import javax.persistence.Column;
import com.haulmont.cuba.core.entity.StandardEntity;
import com.haulmont.chile.core.annotations.NamePattern;

@NamePattern("%s %s|name,phoneNumber")
@Table(name = "WORKSHOP_CLIENT")
@Entity(name = "workshop$Client")
public class Client extends StandardEntity {
    private static final long serialVersionUID = -7748302684971382274L;

    @Column(name = "NAME", nullable = false)
    protected String name;

    @Column(name = "PHONE_NUMBER", nullable = false, unique = true, length = 20)
    protected String phoneNumber;

    @Column(name = "EMAIL", nullable = false, unique = true, length = 50)
    protected String email;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getEmail() {
```

DDL Scripts

1. Click on **DDL Preview** tab of the **Entity designer**
2. Click **OK** to save the ***Client*** entity

This tab illustrates preview of SQL script for corresponding table creation.

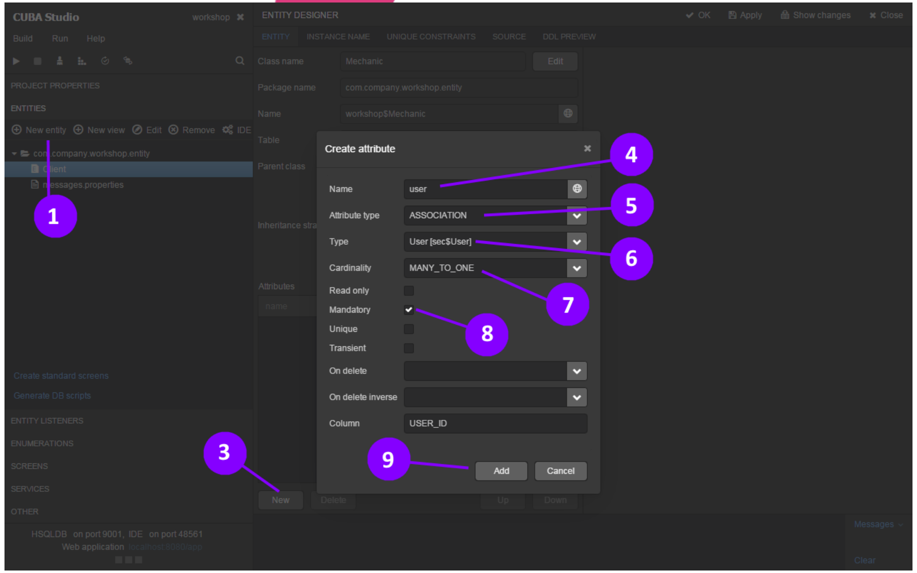


```
create table WORKSHOP_CLIENT (
    ID varchar(36) not null,
    CREATE_TS timestamp,
    CREATED_BY varchar(50),
    VERSION integer,
    UPDATE_TS timestamp,
    UPDATED_BY varchar(50),
    DELETE_TS timestamp,
    DELETED_BY varchar(50),
    --
    NAME varchar(255) not null,
    PHONE_NUMBER varchar(20) not null,
    EMAIL varchar(50) not null,
    --
    primary key (ID)
)^

-- unique indexes

create unique index IDX_WORKSHOP_CLIENT_UNIQ_PHONE_NUMBER on WORKSHOP_CLIENT (PHONE_NUMBER) ^
create unique index IDX_WORKSHOP_CLIENT_UNIQ_EMAIL on WORKSHOP_CLIENT (EMAIL) ^
```

Mechanic entity

1. Click **New entity**
2. Input ***Mechanic*** as entity name and click **OK**
3. Create **New** attribute
4. Set attribute **name** to ***user***
5. Set **Attribute type**: ***ASSOCIATION***
6. Set **Type**: ***User [sec$User]***
7. Set **Cardinality**: ***MANY_TO_ONE***
8. Select **Mandatory** checkbox
9. Click **Add**

The ***User*** entity is a standard entity used to operate with system users in the CUBA Platform.



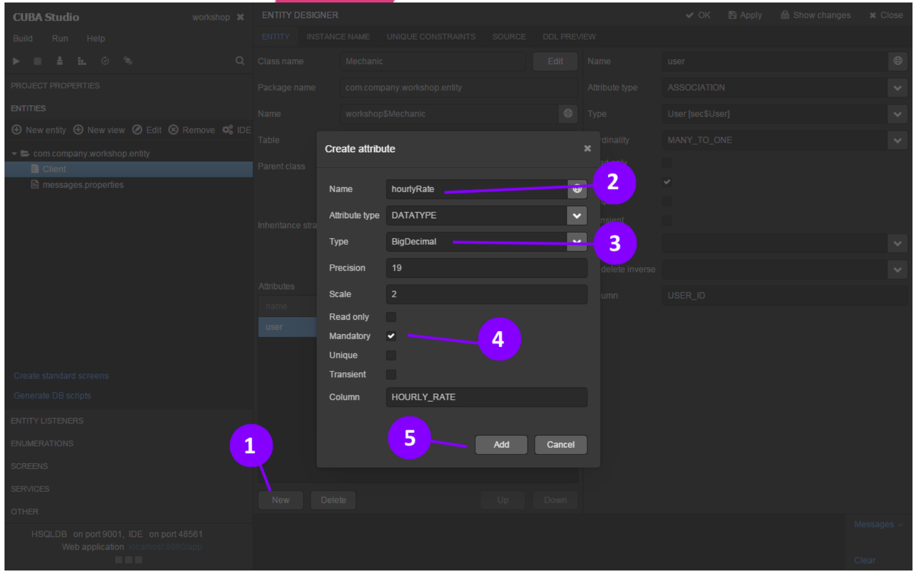
HSQLDB on port 9001, IDE on port 48561
Web application localhost:8080/app

Class name	Mechanic
Package name	com.company.workshop.entity
Name	workshop$Mechanic

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

Mechanic entity — hourlyRate attribute

1. Click **New** to create attribute
2. Set **Name**: ***hourlyRate***
3. Set **Type**: ***BigDecimal***
4. Select **Mandatory** checkbox
5. Click the **Add** button



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

Name	user
Attribute type	ASSOCIATION
Type	User [sec$User]
Cardinality	MANY_TO_ONE
Column	USER_ID

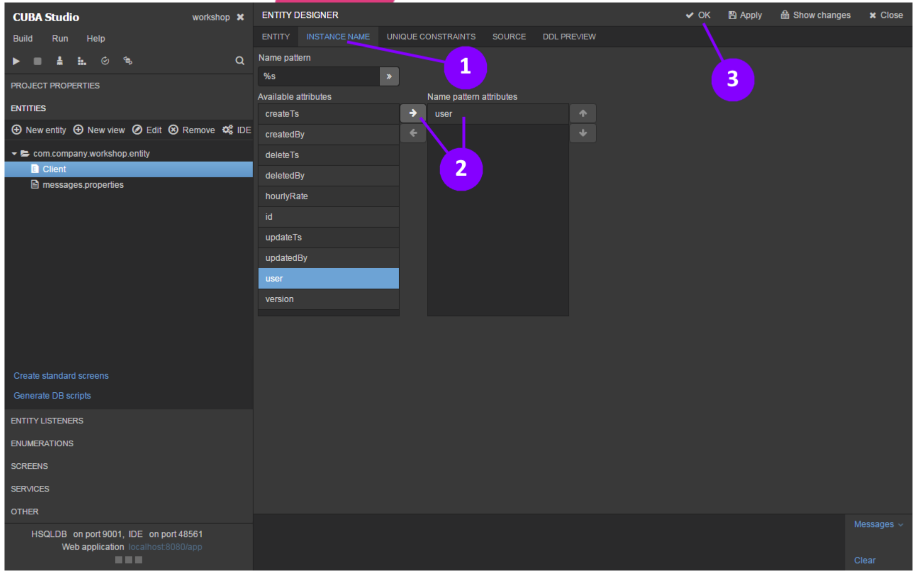
Create attribute

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

Add Cancel

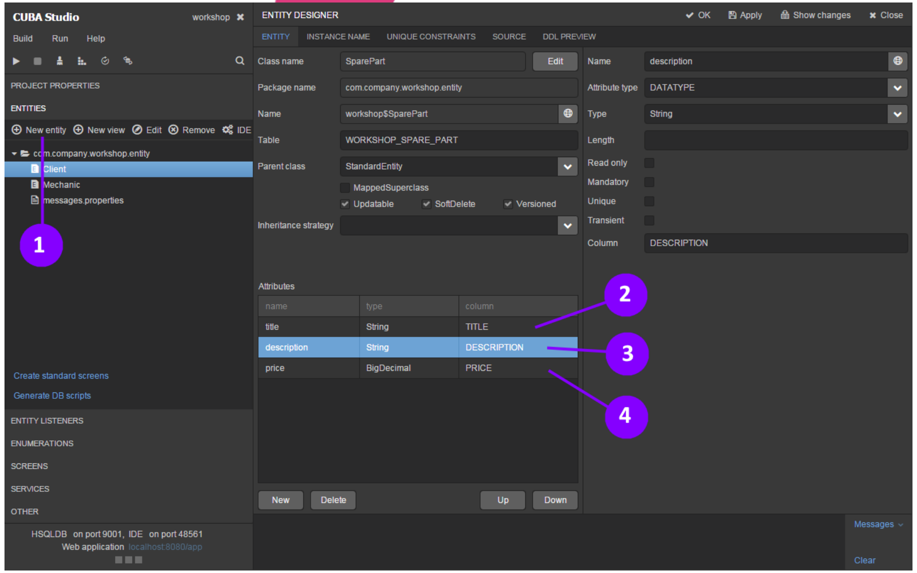
Mechanic entity — instance name

1. Go to the **Instance name** tab
2. Select ***user*** for the **Mechanic's instance name**
3. Save the **Mechanic** entity by clicking **OK**



SparePart entity

1. Create **New entity** with **Class name**: ***SparePart***
2. Add the ***title*** attribute as a **mandatory** and **unique String**
3. Add the ***description*** attribute: **String**; clean up the value of length field, so ***description*** will have unlimited length
4. Add the ***price*** attribute: **mandatory**, **BigDecimal**



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

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	

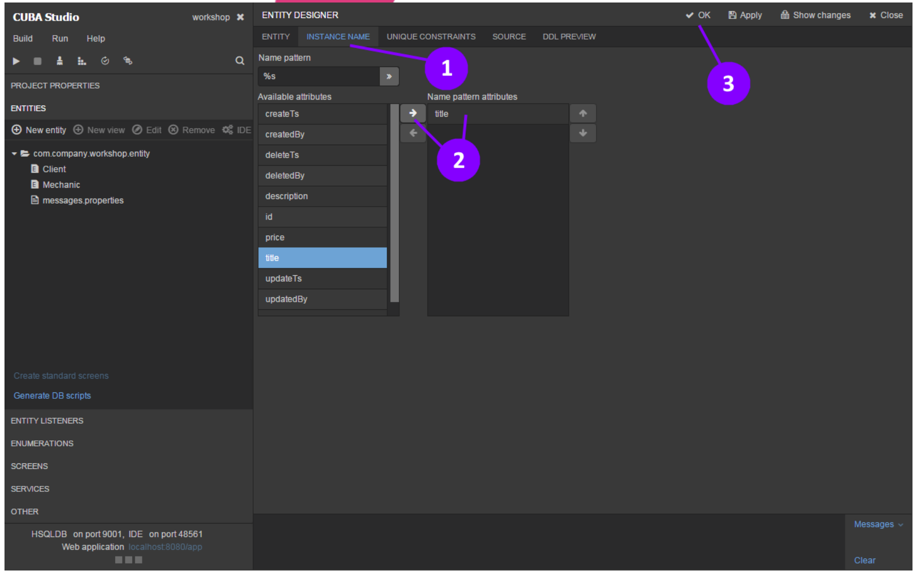
Attributes

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

Name	description
Attribute type	DATATYPE
Type	String
Length	
Read only	
Mandatory	
Unique	
Transient	
Column	DESCRIPTION

SparePart entity — instance name

1. Go to the **Instance name** tab
2. Select the ***title*** attribute for the ***SparePart* instance name**
3. Click **OK** to save the entity

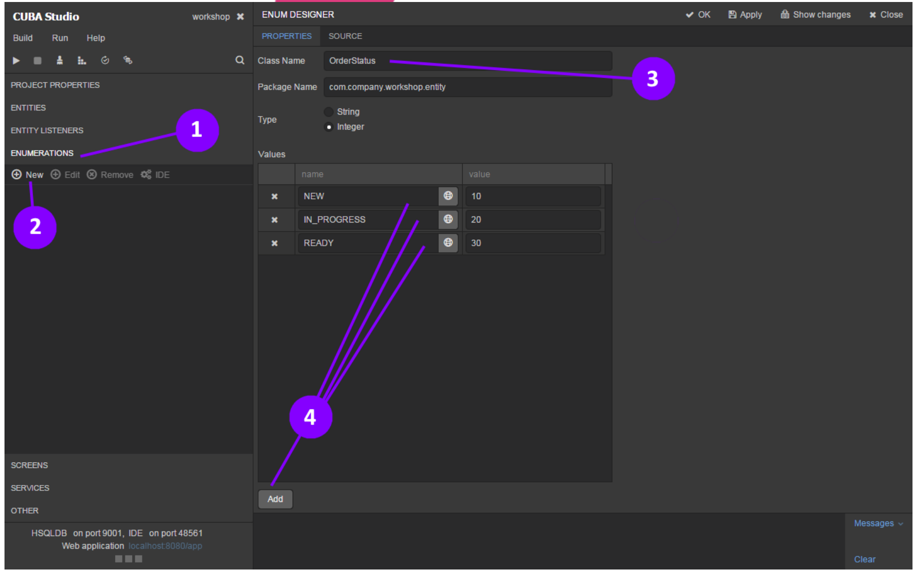


OrderStatus enum

To create the ***Order*** entity we'll need to create the ***OrderStatus*** enum first.

1. Go to the **Enumerations** section in the navigation panel
2. Click **New**
3. Enter **Class Name**: ***OrderStatus***
4. Add values:

NEW 10
IN_PROGRESS 20
READY 30



Class Name: OrderStatus
Package Name: com.company.workshop.entity
Type: String [✓ Integer]

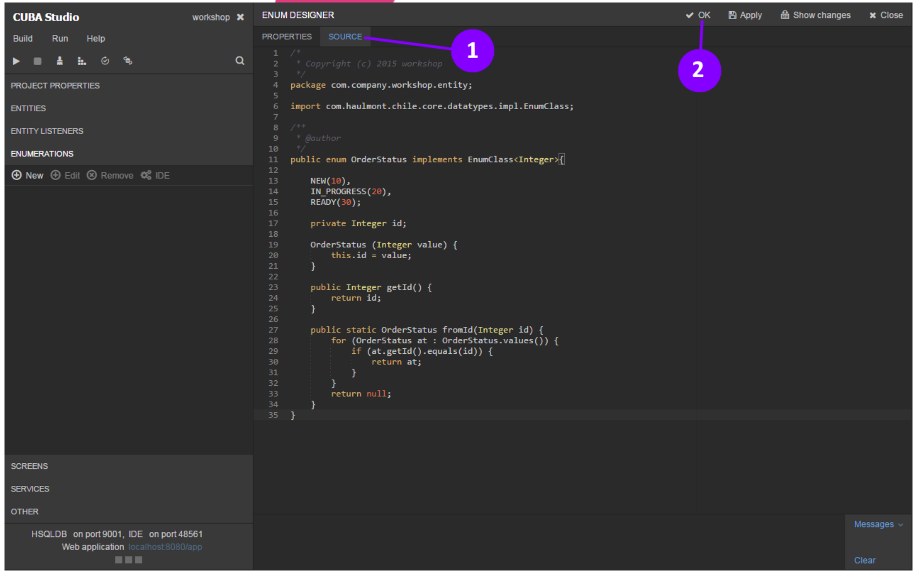
Values

name	value
NEW	10
IN_PROGRESS	20
READY	30

OrderStatus enum — source code

1. Similar to entities, we can check the generated Java code in the **Source** tab
2. Click **OK** to save the enum

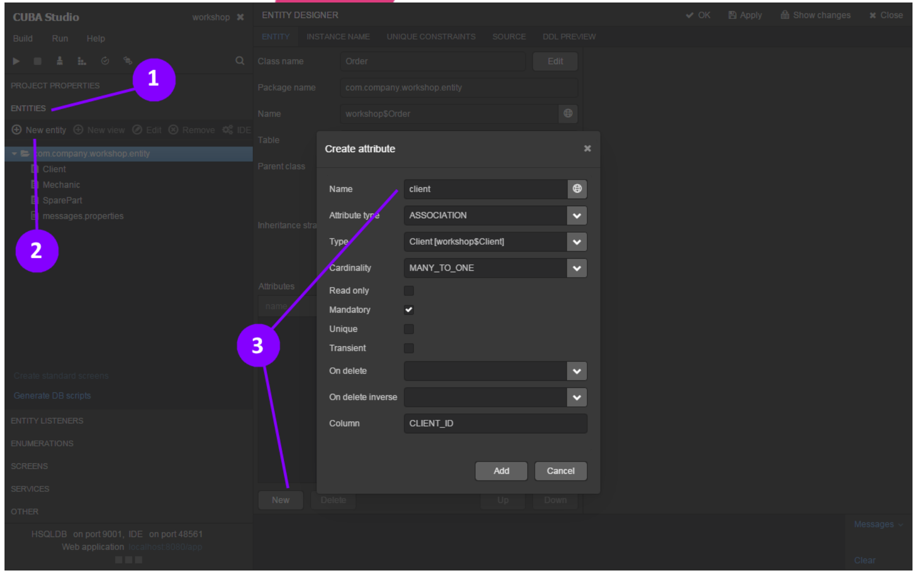
You can change source code of enum manually here and the Studio can read it back from the source to its enum model.



```java
/*
 * Copyright (c) 2015 workshop
 */
package com.company.workshop.entity;

import com.haulmont.chile.core.datatypes.impl.EnumClass;

/**
 * @author
 */
public enum OrderStatus implements EnumClass<Integer>{

    NEW(10),
    IN_PROGRESS(20),
    READY(30);

    private Integer id;

    OrderStatus (Integer value) {
        this.id = value;
    }

    public Integer getId() {
        return id;
    }

    public static OrderStatus fromId(Integer id) {
        for (OrderStatus at : OrderStatus.values()) {
            if (at.getId().equals(id)) {
                return at;
            }
        }
        return null;
    }
}
```

Order entity

1. Go to the **Entities** section of the navigation panel
2. Create **new entity**
3. Set ***Order*** as the **Class name**
4. Add new attribute **named**: ***client***
   Attribute type: ***ASSOCIATION***
   Type: ***Client***
   Cardinality: ***MANY_TO_ONE***
   Mandatory: ***true***
5. Similarly add the **mechanic** attribute with
   Type: *Mechanic*



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

Create attribute

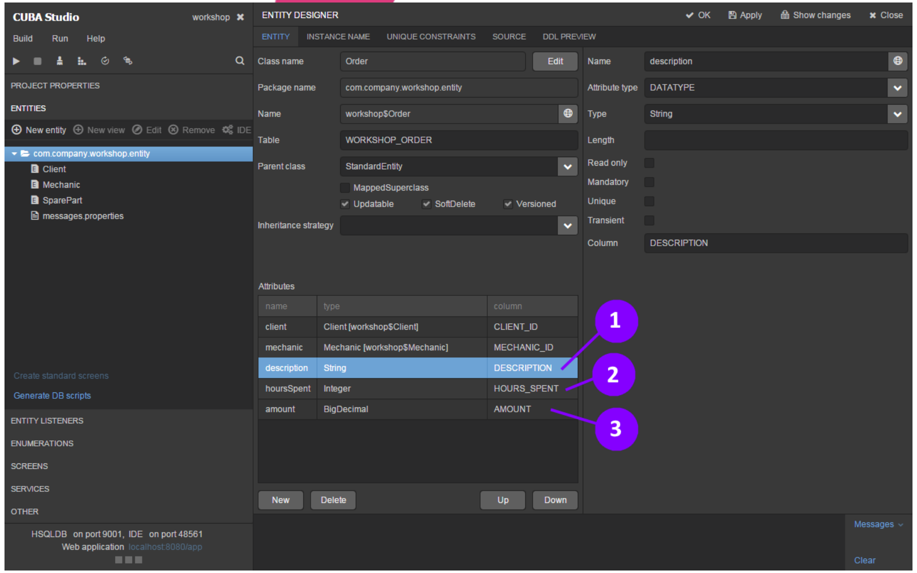
Name: client
Attribute type: ASSOCIATION
Type: Client [workshop$Client]
Cardinality: MANY_TO_ONE
Read only: ☐
Mandatory: ✓
Unique: ☐
Transient: ☐
On delete:
On delete inverse:
Column: CLIENT_ID

Add | Cancel

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

Order entity — description, hoursSpent, amount

1. Add ***description*** attribute: **String**, clean up the value of length field, so ***description*** will have unlimited length
2. Add ***hoursSpent*** attribute: **Integer**
3. Add ***amount*** attribute: **BigDecimal**



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

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

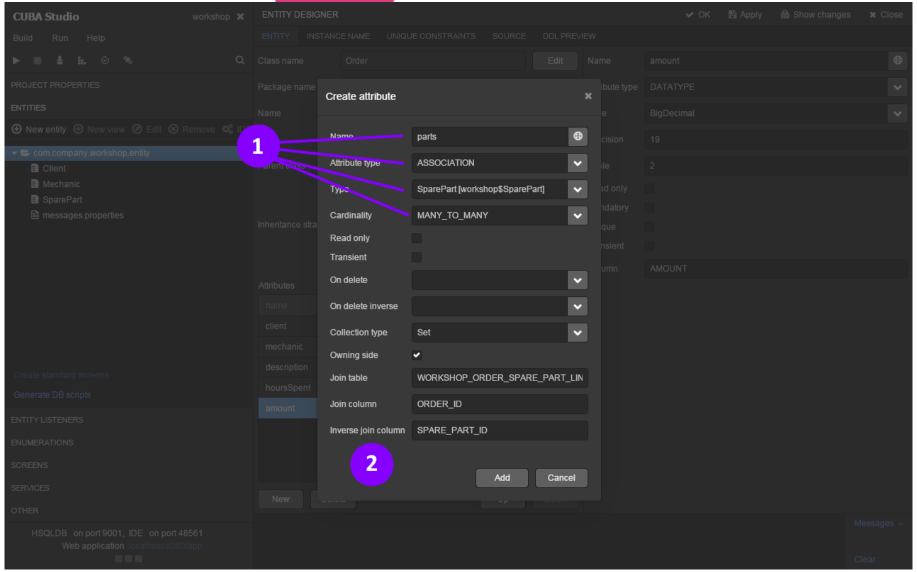
name	type	column
client	Client [workshop$Client]	CLIENT_ID
mechanic	Mechanic [workshop$Mechanic]	MECHANIC_ID
description	String	DESCRIPTION
hoursSpent	Integer	HOURS_SPENT
amount	BigDecimal	AMOUNT

Name	description
Attribute type	DATATYPE
Type	String
Length	
Read only	☐
Mandatory	☐
Unique	☐
Transient	☐
Column	DESCRIPTION

CUBA.platform
www.cuba-platform.com

Order entity — parts attribute

1. Create a **New** attribute: ***parts***
   Attribute type: ***ASSOCIATION***
   Type: ***SparePart***
   Cardinality: ***MANY_TO_MANY***
2. Click on the **Add** button
3. The Studio will offer to create a reverse attribute from the ***SparePart*** entity to link it to ***Order***, just click **No**

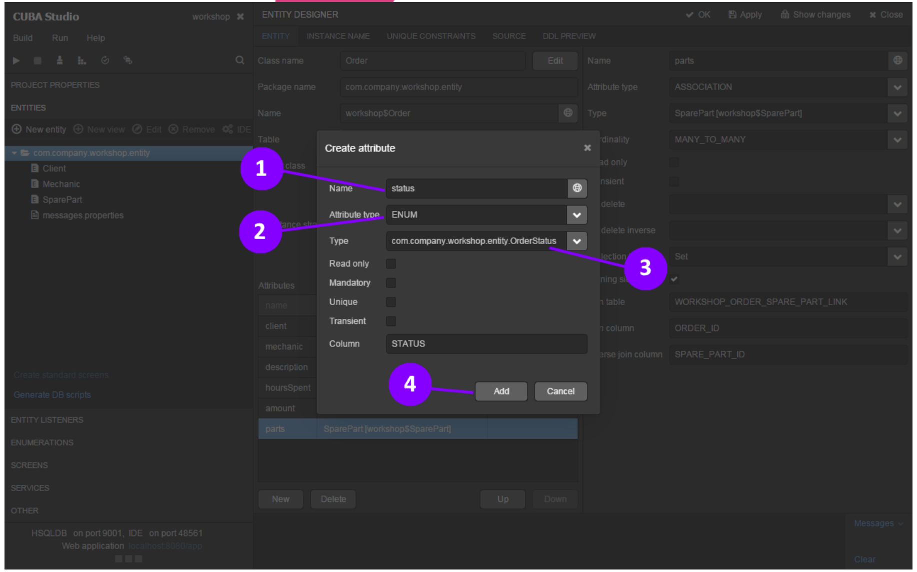


Create attribute

Name	parts
Attribute type	ASSOCIATION
Type	SparePart [workshop$SparePart]
Cardinality	MANY_TO_MANY
Read only	
Transient	
On delete	
On delete inverse	
Collection type	Set
Owning side	✓
Join table	WORKSHOP_ORDER_SPARE_PART_LIN
Join column	ORDER_ID
Inverse join column	SPARE_PART_ID

Order entity — status attribute

1. Create **New** attribute: ***status***
2. Set **Attribute type**: ***ENUM***
3. Set **Type**: ***OrderStatus***
4. Click **Add**



Create attribute

Name	status
Attribute type	ENUM
Type	com.company.workshop.entity.OrderStatus
Read only	☐
Mandatory	☐
Unique	☐
Transient	☐
Column	STATUS

Add Cancel

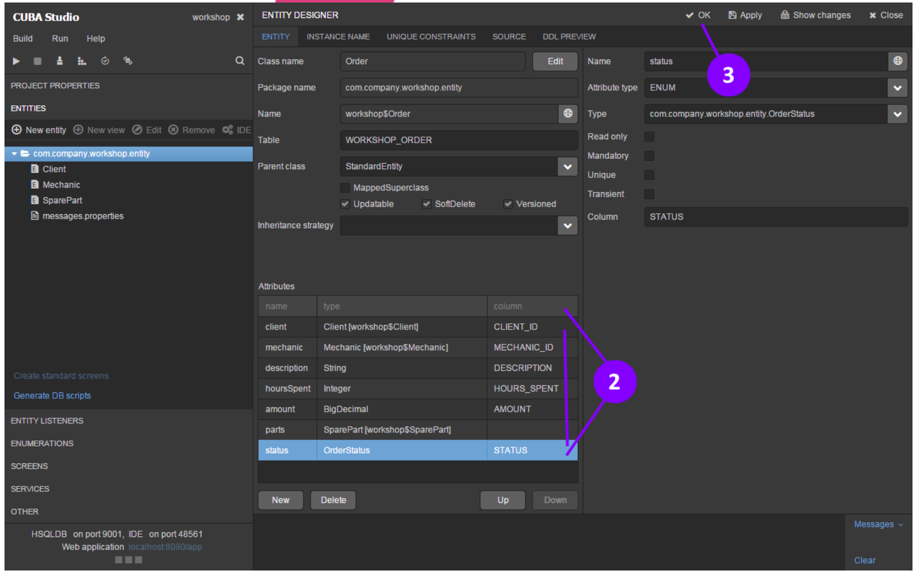
Class name	Order	Edit
Package name	com.company.workshop.entity	
Name	workshop$Order	

Name	parts
Attribute type	ASSOCIATION
Type	SparePart [workshop$SparePart]
dinality	MANY_TO_MANY
lection	Set
n table	WORKSHOP_ORDER_SPARE_PART_LINK
n column	ORDER_ID
erse join column	SPARE_PART_ID

name	
client	
mechanic	
description	
hoursSpent	
amount	
parts	SparePart [workshop$SparePart]

Order entity — done

1. Set **Instance name** for the ***Order*** entity to its ***description*** *attribute*
2. Check the attributes list of the ***Order*** entity: ***client***, ***mechanic***, ***description***, ***hoursSpent***, ***amount***, ***parts***, ***status***
3. Click **OK** to save the entity



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
description	String	DESCRIPTION
hoursSpent	Integer	HOURS_SPENT
amount	BigDecimal	AMOUNT
parts	SparePart [workshop$SparePart]	
status	OrderStatus	STATUS

Name	status
Attribute type	ENUM
Type	com.company.workshop.entity.OrderStatus
Read only	☐
Mandatory	☐
Unique	☐
Transient	☐
Column	STATUS

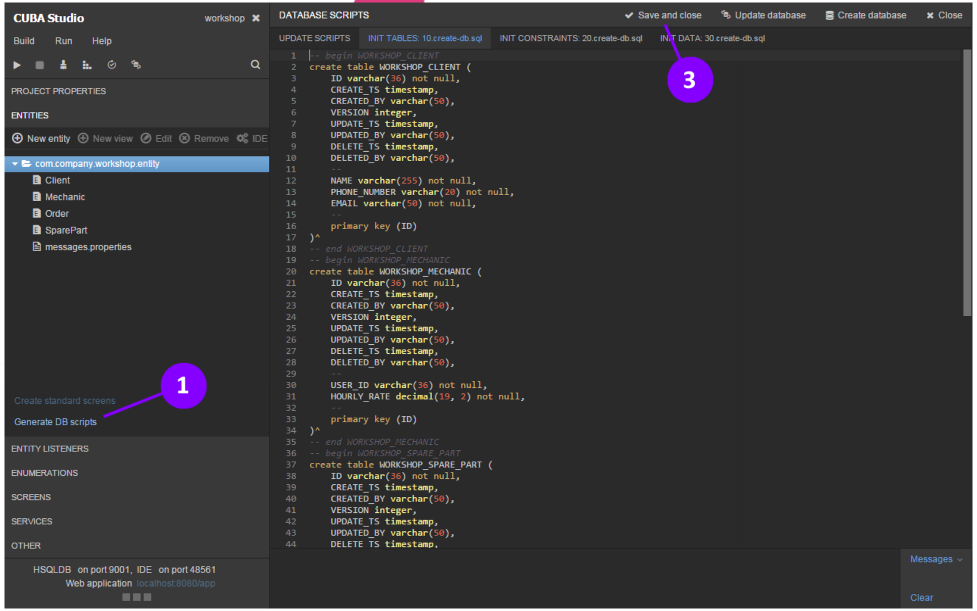
HSQLDB on port 9001, IDE on port 48561
Web application localhost:8080/app

CUBA.platform
www.cuba-platform.com

Database

Generate DB scripts

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

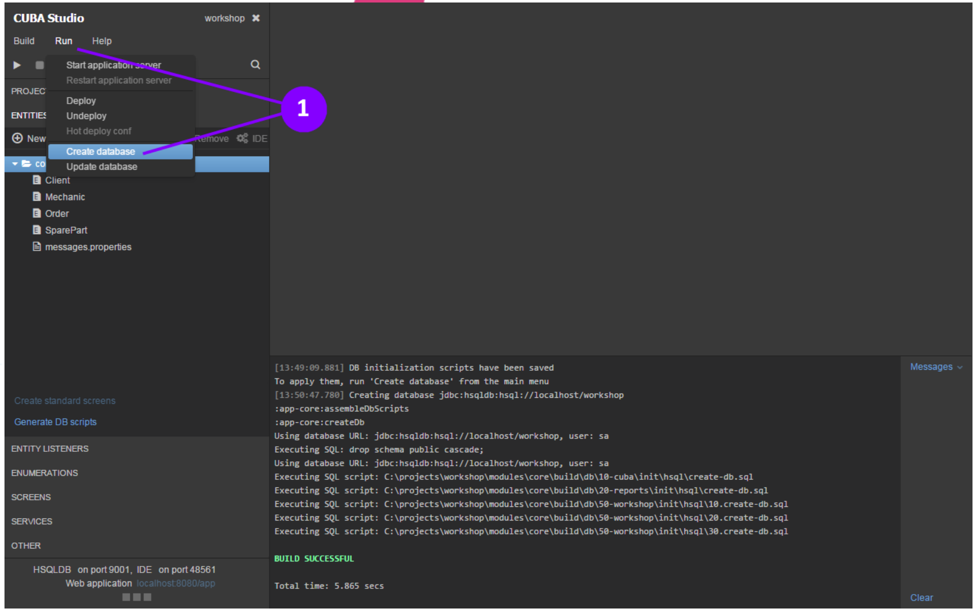


```sql
-- begin WORKSHOP_CLIENT
create table WORKSHOP_CLIENT (
    ID varchar(36) not null,
    CREATE_TS timestamp,
    CREATED_BY varchar(50),
    VERSION integer,
    UPDATE_TS timestamp,
    UPDATED_BY varchar(50),
    DELETE_TS timestamp,
    DELETED_BY varchar(50),
    --
    NAME varchar(255) 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,
    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,
    CREATED_BY varchar(50),
    VERSION integer,
    UPDATE_TS timestamp,
    UPDATED_BY varchar(50),
    DELETE_TS timestamp,
```

Create database

1. Invoke the **Run — Create database** action from the menu to create a database
2. The CUBA Studio warns us that the old DB will be deleted, click **OK**

The Studio outputs process stages to the log. When **Build Successful** message is shown, our DB is created.



```
[13:49:09.881] DB initialization scripts have been saved
To apply them, run 'Create database' from the main menu
[13:50:47.780] Creating database jdbc:hsqldb:hsql://localhost/workshop
:app-core:assembleDbScripts
:app-core:createDb
Using database URL: jdbc:hsqldb:hsql://localhost/workshop, user: sa
Executing SQL: drop schema public cascade;
Using database URL: jdbc:hsqldb:hsql://localhost/workshop, user: sa
Executing SQL script: C:\projects\workshop\modules\core\build\db\10-cuba\init\hsql\create-db.sql
Executing SQL script: C:\projects\workshop\modules\core\build\db\20-reports\init\hsql\create-db.sql
Executing SQL script: C:\projects\workshop\modules\core\build\db\50-workshop\init\hsql\10.create-db.sql
Executing SQL script: C:\projects\workshop\modules\core\build\db\50-workshop\init\hsql\20.create-db.sql
Executing SQL script: C:\projects\workshop\modules\core\build\db\50-workshop\init\hsql\30.create-db.sql

BUILD SUCCESSFUL

Total time: 5.865 secs
```

CUBA.platform
www.cuba-platform.com

User Interface

Screens scaffolding

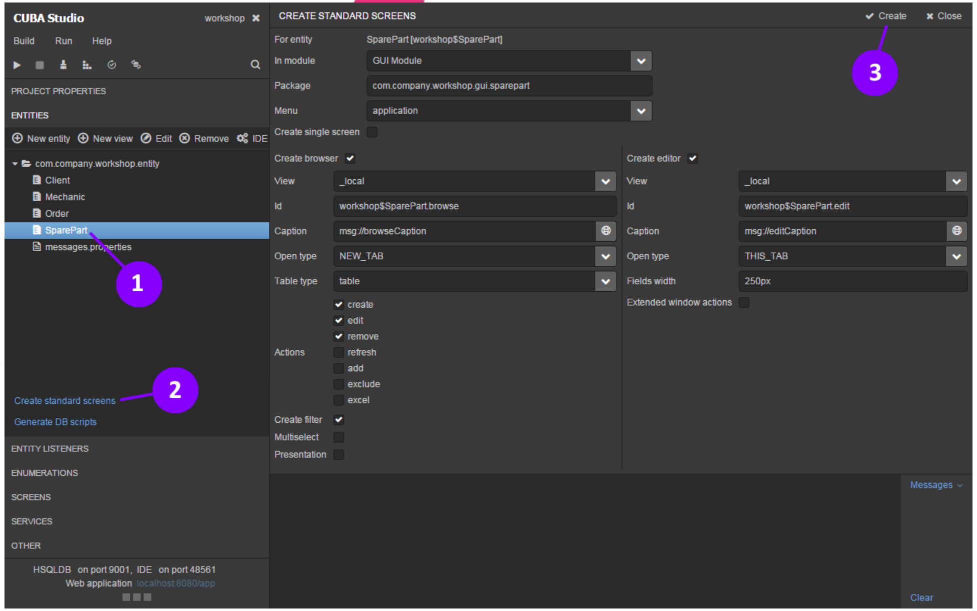
Now let's create standard browser and editor screens for the ***SparePart*** entity.

1. Select ***SparePart*** in the navigation panel
2. Click on the **Create standard screens** link
3. Click **Create**

On this screen we can specify where to place the screens and which menu item will be used to open the browser screen.

The following terminology is used:

- Browser screen — screen with list of records
- Editor screen — simple edit form for record



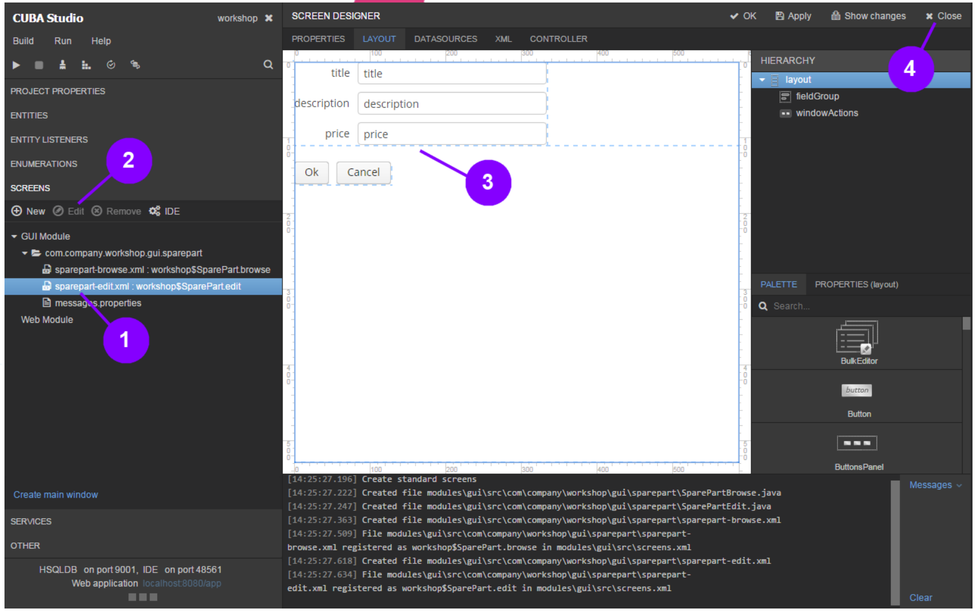
For entity	SparePart [workshop$SparePart]
In module	GUI Module
Package	com.company.workshop.gui.sparepart
Menu	application
Create single screen	☐

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	☐		

Screen designer

The Studio has generated 2 screens. Let’s have a look at **sparepart-edit.xml**.

1. Select **sparepart-edit.xml** in the **Screens** section
2. Click **Edit**
3. The CUBA Studio features a built-in WYSIWIG screens editor to speed up UI development
4. Click **Close**



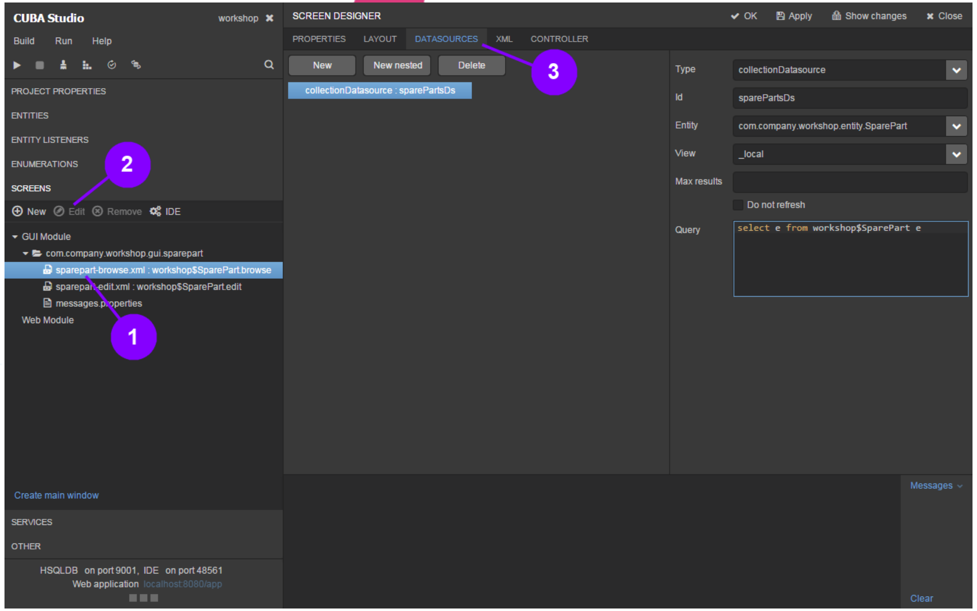
```
[14:25:27.196] Create standard screens
[14:25:27.222] Created file modules\gui\src\com\company\workshop\gui\sparepart\SparePartBrowse.java
[14:25:27.247] Created file modules\gui\src\com\company\workshop\gui\sparepart\SparePartEdit.java
[14:25:27.363] Created file modules\gui\src\com\company\workshop\gui\sparepart\sparepart-browse.xml
[14:25:27.509] File modules\gui\src\com\company\workshop\gui\sparepart\sparepart-
browse.xml registered as workshop$SparePart.browse in modules\gui\src\screens.xml
[14:25:27.618] Created file modules\gui\src\com\company\workshop\gui\sparepart\sparepart-edit.xml
[14:25:27.634] File modules\gui\src\com\company\workshop\gui\sparepart\sparepart-
edit.xml registered as workshop$SparePart.edit in modules\gui\src\screens.xml
```

Data binding

Components are connected with data sources, which are configurable from the **Datasources** tab.

1. Select **sparepart-browse.xml**
2. Click **Edit**
3. Go to the **Datasources** tab

Datasources use JPQL queries to load data.

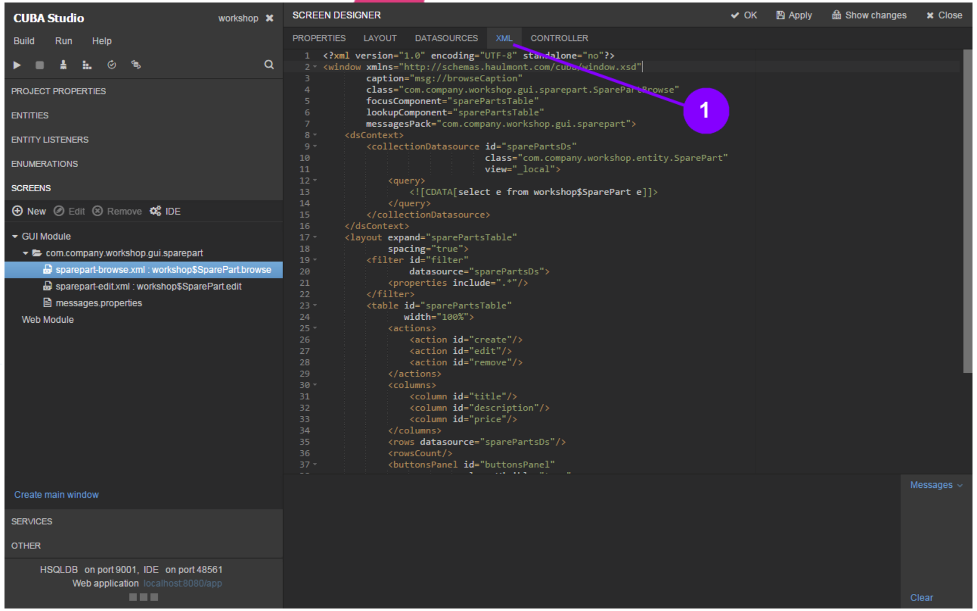


Type	collectionDatasource
Id	sparePartsDs
Entity	com.company.workshop.entity.SparePart
View	_local
Max results	
	Do not refresh
Query	select e from workshop$SparePart e

Declarative UI definition

1. UI is described declaratively using XML, we can see an example of the descriptor in the **XML** tab

The XML view is synchronized with the graphical design, and if we make changes in XML, then the graphical view will be updated and vice versa.

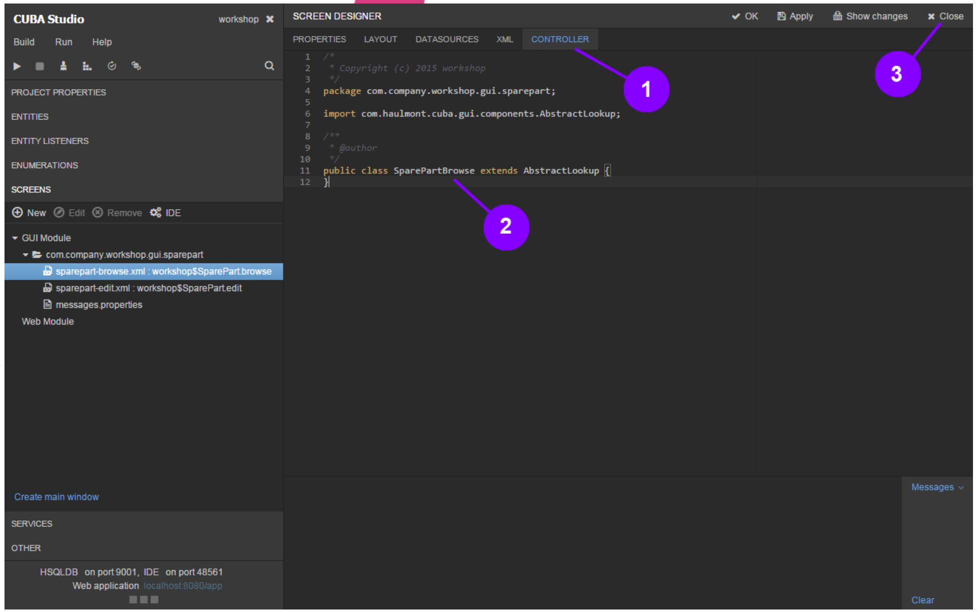


```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<window xmlns="http://schemas.haulmont.com/cuba/window.xsd"
        caption="msg://browseCaption"
        class="com.company.workshop.gui.sparepart.SparePartBrowse"
        focusComponent="sparePartsTable"
        lookupComponent="sparePartsTable"
        messagesPack="com.company.workshop.gui.sparepart">
    <dsContext>
        <collectionDatasource id="sparePartsDs"
                              class="com.company.workshop.entity.SparePart"
                              view="_local">
            <query>
                <![CDATA[select e from workshop$SparePart e]]>
            </query>
        </collectionDatasource>
    </dsContext>
    <layout expand="sparePartsTable"
            spacing="true">
        <filter id="filter"
                datasource="sparePartsDs">
            <properties include=".*"/>
        </filter>
        <table id="sparePartsTable"
               width="100%">
            <actions>
                <action id="create"/>
                <action id="edit"/>
                <action id="remove"/>
            </actions>
            <columns>
                <column id="title"/>
                <column id="description"/>
                <column id="price"/>
            </columns>
            <rows datasource="sparePartsDs"/>
            <rowsCount/>
            <buttonsPanel id="buttonsPanel"
```

CUBA.platform
www.cuba-platform.com

Screen controller

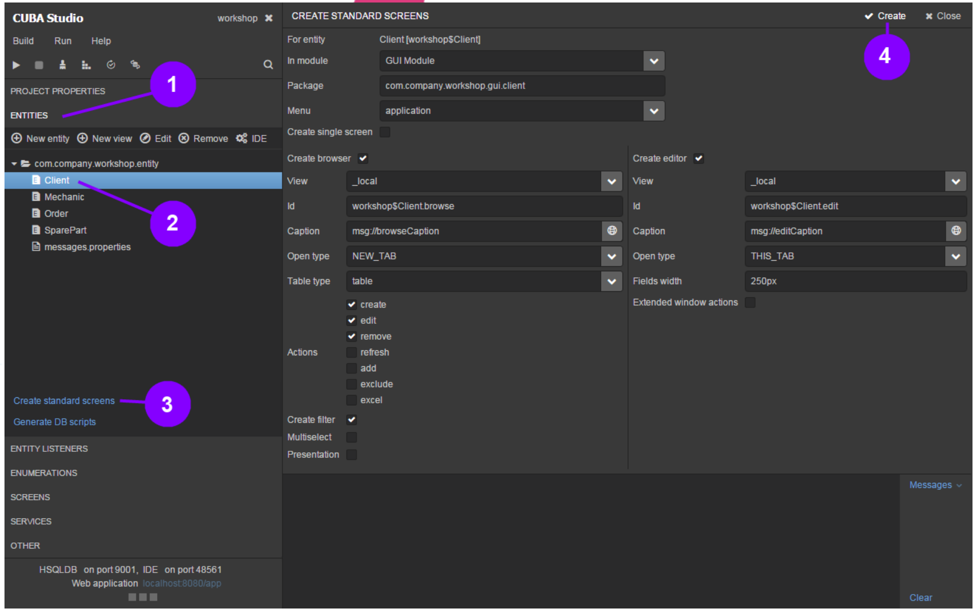
1. Go to the **Controller** tab
2. Apart from XML, the Studio creates a controller for each screen, which is a Java class that implements the logic and handling component events
3. Click **Close**



```java
/*
 * Copyright (c) 2015 workshop
 */
package com.company.workshop.gui.sparepart;

import com.haulmont.cuba.gui.components.AbstractLookup;

/**
 * @author
 */
public class SparePartBrowse extends AbstractLookup {
}
```

Generate screens for Client entity

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



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

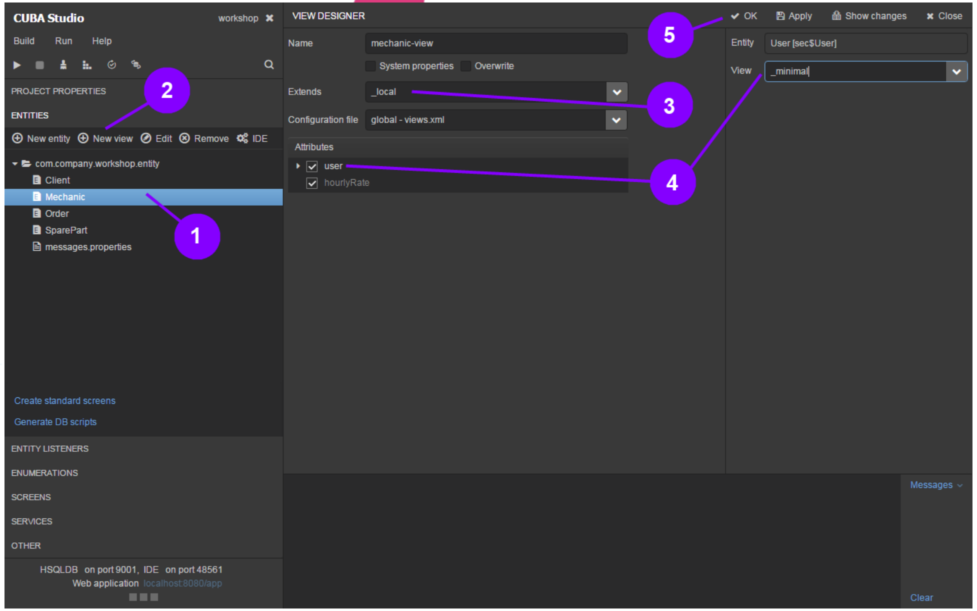
Create browser: ✓
View: _local
Id: workshop$Client.browse
Caption: msg://browseCaption
Open type: NEW_TAB
Table type: table
Actions: ✓ create, ✓ edit, ✓ remove, ☐ refresh, ☐ add, ☐ exclude, ☐ excel
Create filter: ✓
Multiselect: ☐
Presentation: ☐

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

View. Loading of entity graphs from DB

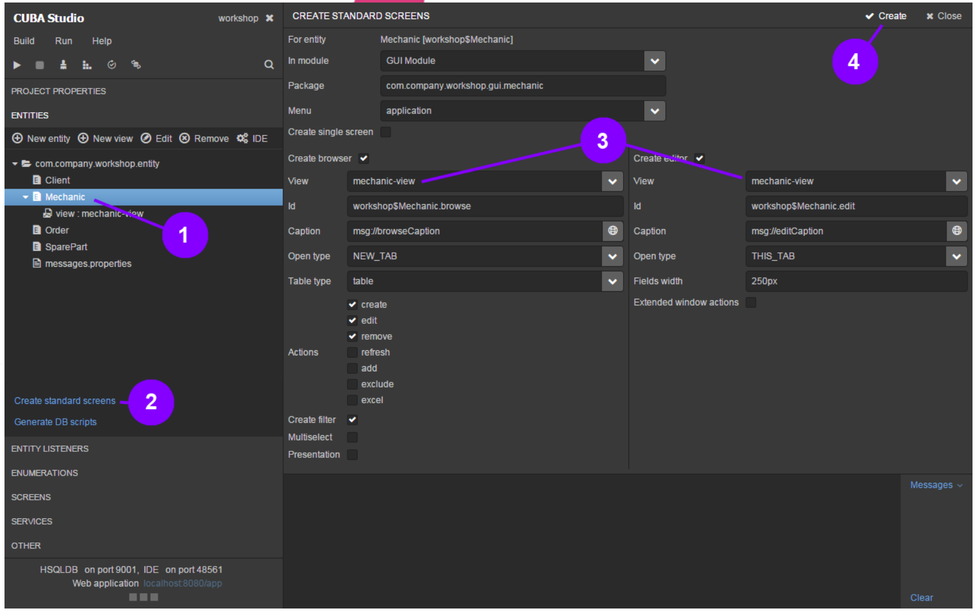
The ***Mechanic*** entity is linked to ***User***. So, we need to load related ***User*** entity to display it in the browser and editor screens. In CUBA, this is done via special object — **View**, which describes what entity attributes should be loaded from the database. Let's create a view for the ***Mechanic*** entity, which will include ***User***.

1. Select the ***Mechanic*** entity
2. Click **New view**
3. Choose **Extends view**: ***_local***, as we want to include all local attributes
4. Select the ***user*** attribute, specify ***_minimal*** view for this attribute **_minimal** view includes only attributes that are specified in the **Instance Name** of an entity
5. Click **OK** to save the view



Generate screens for Mechanic

1. Select the ***Mechanic*** entity
2. Click **Create standard screens**
3. Choose ***mechanic-view*** for browser and editor screens
4. Click **Create**



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

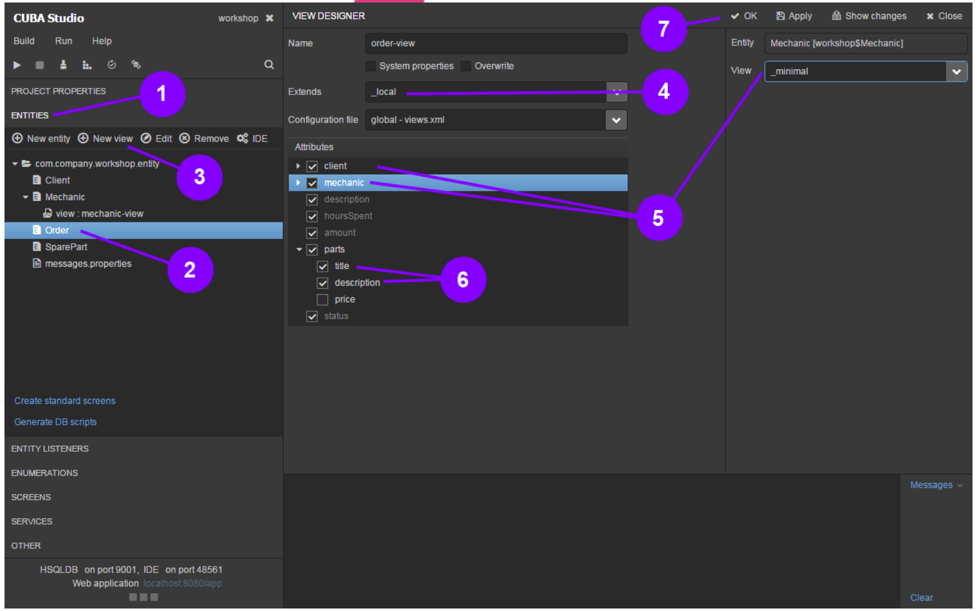
Create browser	✓	Create editor	✓
View	mechanic-view	View	mechanic-view
Id	workshop$Mechanic.browse	Id	workshop$Mechanic.edit
Caption	msg://browseCaption	Caption	msg://editCaption
Open type	NEW_TAB	Open type	THIS_TAB
Table type	table	Fields width	250px
Actions	✓ create, ✓ edit, ✓ remove, ☐ refresh, ☐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

CUBA.platform
www.cuba-platform.com

View for Order browser and editor

Now we need to create screens for the ***Order*** entity. We'll also need to create a special view.

1. Open the **Entities** section of the navigation panel
2. Select the ***Order*** entity
3. Click **New view**
4. Set **Extends** to ***_local*** to include all local properties
5. Tick ***client***, ***mechanic*** and select the ***_minimal*** view for them
6. Tick ***title*** and ***price*** for ***parts***
7. Click **OK** to save the view



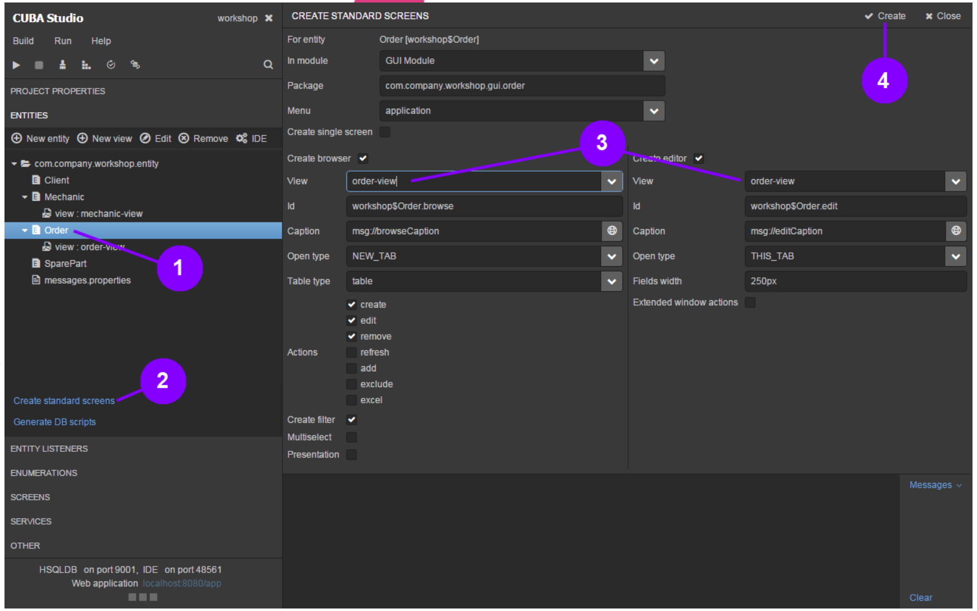
Name: order-view
System properties
Overwrite
Extends: _local
Configuration file: global - views.xml

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

Entity: Mechanic [workshop$Mechanic]
View: _minimal

Generate screens for the Order entity

1. Select the ***Order*** entity
2. Click **Create standard screens**
3. Choose **order-view** for browser and editor screens
4. Click **Create**



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

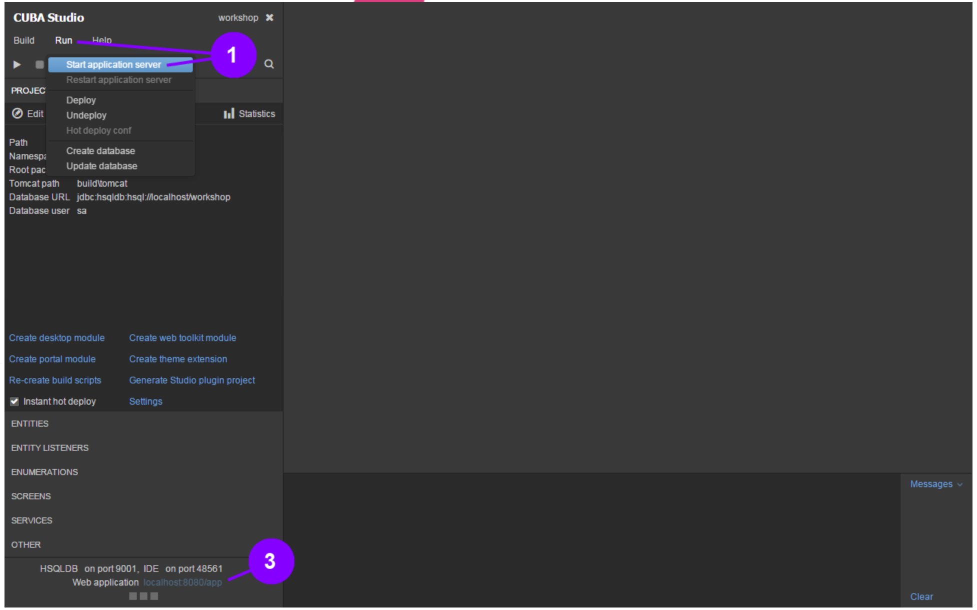
For entity	Order [workshop$Order]
In module	GUI Module
Package	com.company.workshop.gui.order
Menu	application
Create single screen	☐

Create browser	✓	Create editor	✓
View	order-view	View	order-view
Id	workshop$Order.browse	Id	workshop$Order.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	☐		

Let's test it

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.
2. Studio will deploy a local Tomcat instance in the project subdirectory, deploy the compiled application there and launch it.
3. Open the application by clicking a link in the bottom part of the Studio.



Path
Namespa
Root pac
Tomcat path build\tomcat
Database URL jdbc:hsqldb:hsql://localhost/workshop
Database user sa

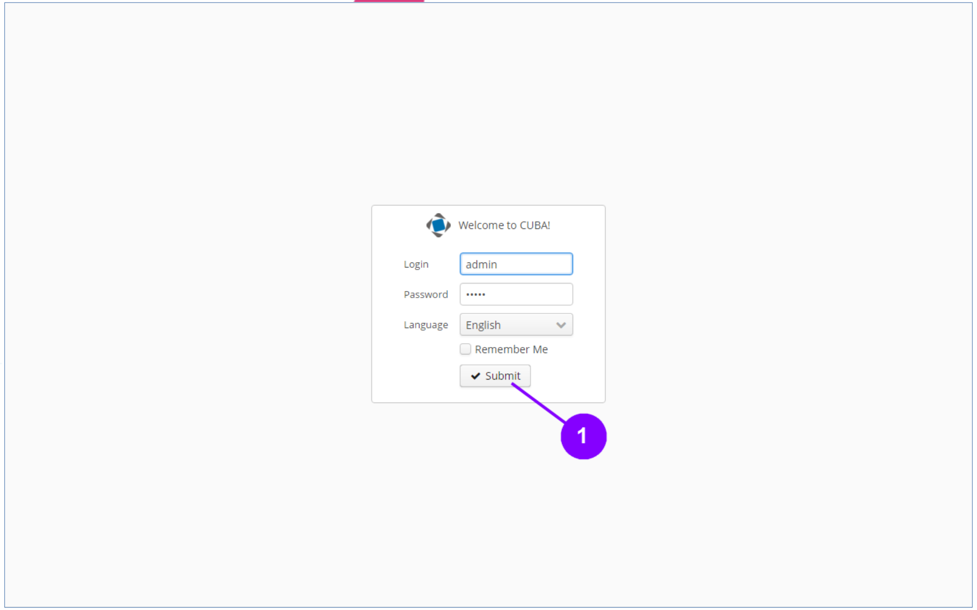
HSQLDB on port 9001, IDE on port 48561
Web application localhost:8080/app

First launch and CRUD

Login screen

The system login screen has appeared. This is a standard CUBA screen, which can be customized, as everything in CUBA, to meet specific requirements.

1. Click **Submit** to login

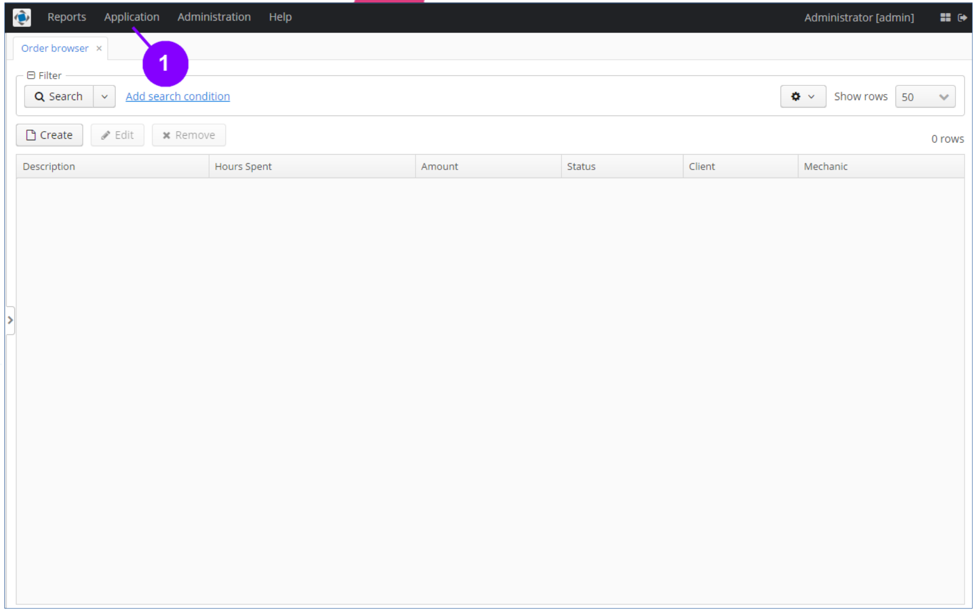


Order browser

Since we have not changed the application menu, our items are displayed by default under the **Application** menu.

1. Open **Application — Orders** from the menu

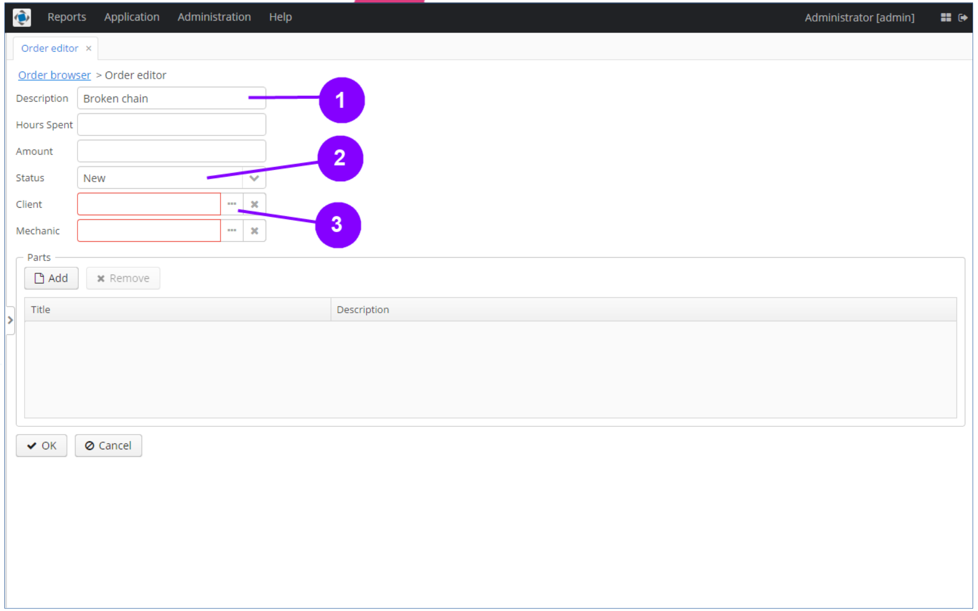
This is a standard browser screen with a filter on top and a table below.



Description	Hours Spent	Amount	Status	Client	Mechanic

Order edit screen

1. Click **Create** and enter the ***description***
2. Select Status: **New**
3. Click button **[…]** to select a ***client*** for the order



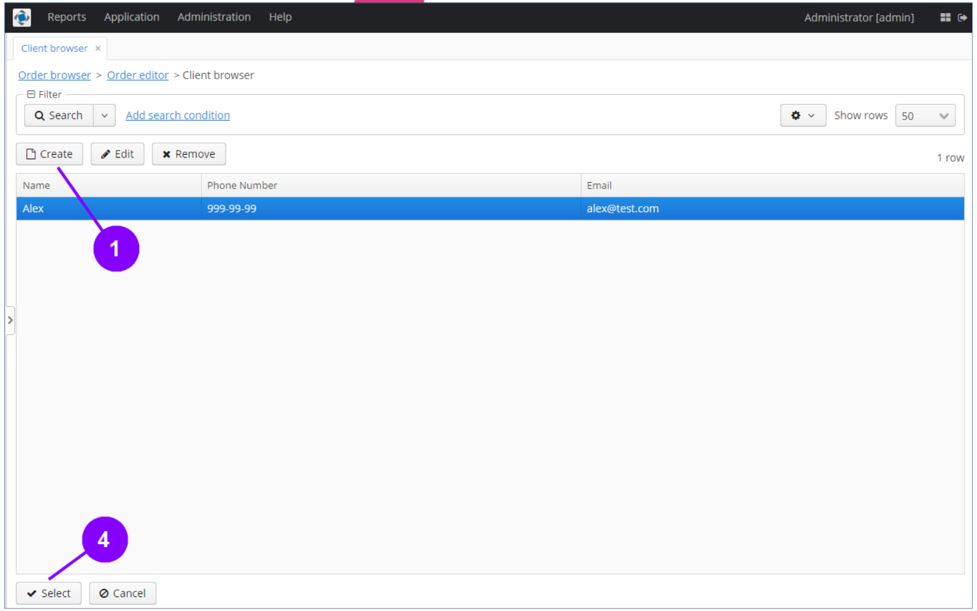
Description	Broken chain
Hours Spent	
Amount	
Status	New
Client	
Mechanic	

Parts

Client browser

So far we don’t have any clients. Let's create one.

1. Click **Create**
2. Fill attributes of the new client
   Name: ***Alex***
   Phone number: **999-99-99**
   Email: ***alex@test.com***
3. Click **OK**
4. Click **Select** to set client to the order



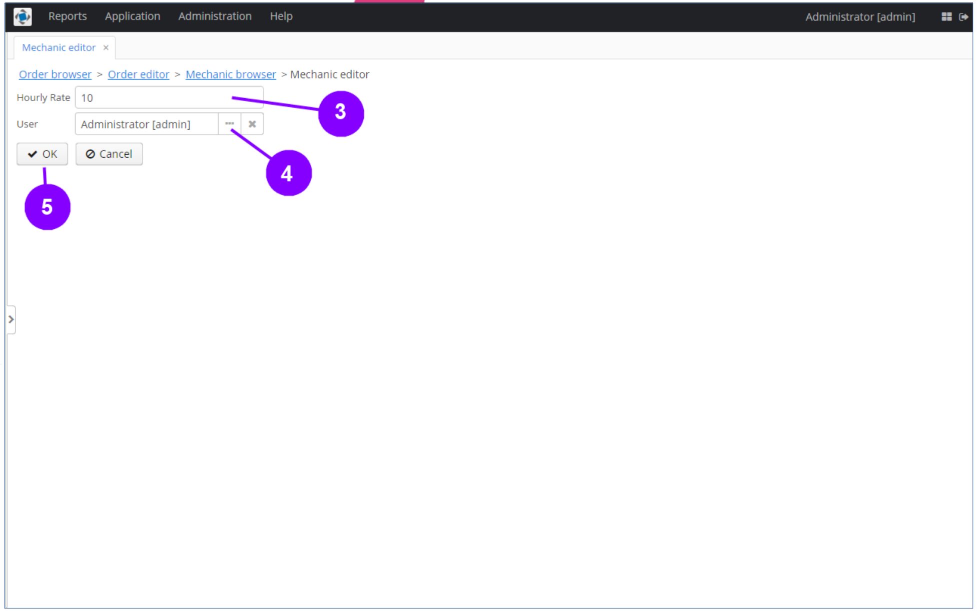
Name	Phone Number	Email
Alex	999-99-99	alex@test.com

Assign mechanic for the order

You are now back to the ***Order* editor** screen

1. Click button **[…]** at the right of the **mechanic field** in the **Order editor**
2. Click **Create** to add a new mechanic
3. Enter **hourly rate**
4. Select ***admin*** user for this mechanic
5. Click **OK**
6. Select mechanic for the order

You can go back to any of opened screens using navigation at the top of screen.



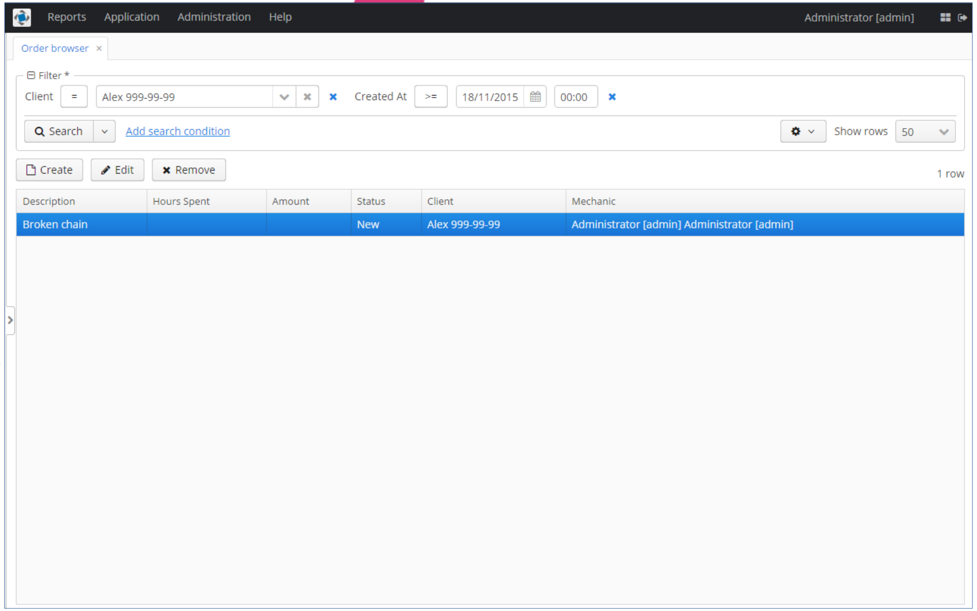
CRUD application

1. Click **OK** to save the order

This is a small working CRUD application that writes data to the database and allows you to simply keep track of orders.

We can search for orders using our Filter.

Table component enables us to hide and change width of columns. Also our table is sortable.



Client = Alex 999-99-99 Created At >= 18/11/2015 00:00

Show rows 50

1 row

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

CUBA.platform
www.cuba-platform.com

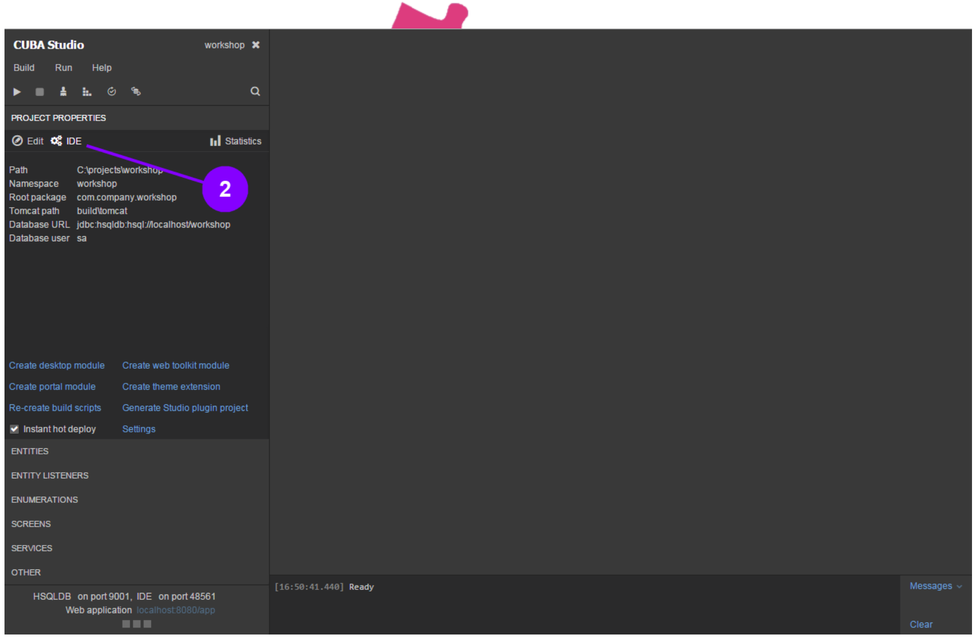
Integration with IDE and project structure

Go to the IDE

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
2. Go to the Studio and click the **IDE** button in the **Project properties** section

The project will come up in the IDE.



Path	C:\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

```
[16:50:41.440] Ready
```

Project structure

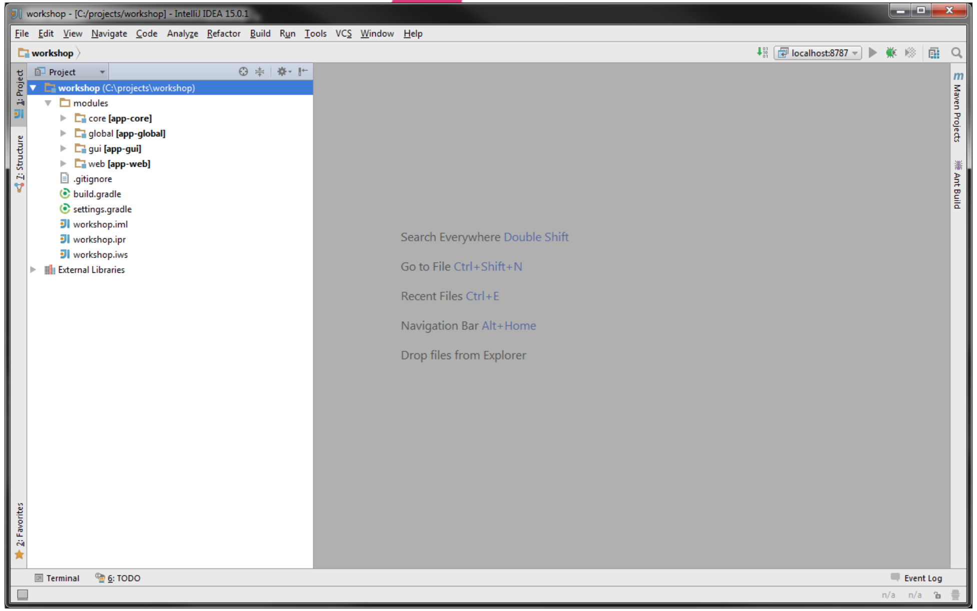
By default any project consists of 4 modules: **global**, **core**, **web**, **gui**.

The **global** module contains data model classes, **core** - middle tier services, **gui** - screens and components, **web** - web client-specific code.

You can have other clients in your project, such as a desktop application or a web portal, which will be placed in separate modules.

The project root directory contains the application build scripts.

Applications are built using Gradle.


workshop - [C:/projects/workshop] - IntelliJ IDEA 15.0.1
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
localhost:8787
Project
Terminal
6: TODO
Event Log

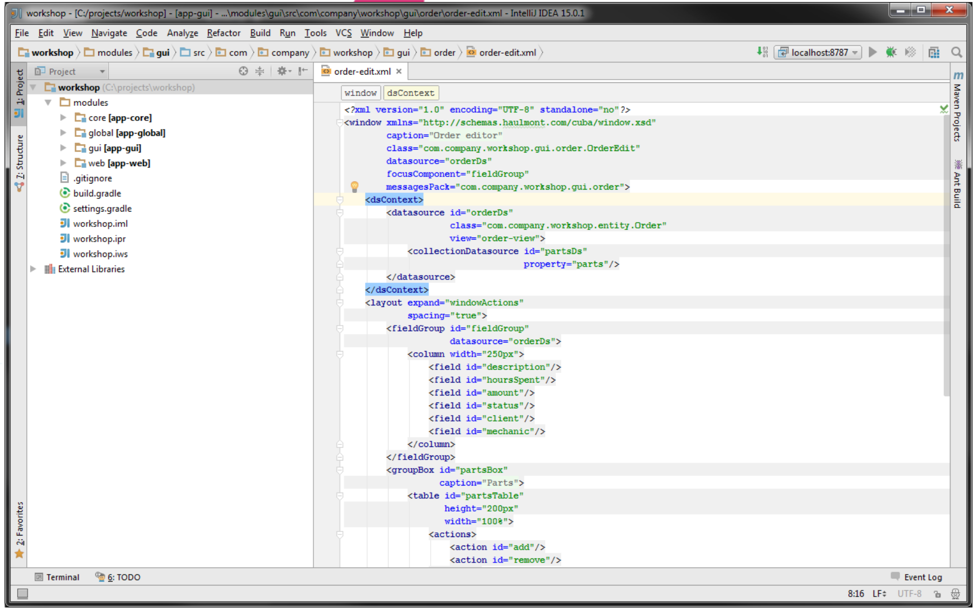
```
workshop (C:\projects\workshop)
  modules
    core [app-core]
    global [app-global]
    gui [app-gui]
    web [app-web]
  .gitignore
  build.gradle
  settings.gradle
  workshop.iml
  workshop.ipr
  workshop.iws
External Libraries
```

Search Everywhere Double Shift
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files from Explorer

CUBA Studio IDE integration

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

IntelliJ IDEA will open the **order-edit.xml** file.
We can edit any file of the project manually using IntelliJ IDEA (or your favorite IDE).


workshop - [C:/projects/workshop] - [app-gui] - ...\modules\gui\src\com\company\workshop\gui\order\order-edit.xml - IntelliJ IDEA 15.0.1
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
workshop modules gui src com company workshop gui order order-edit.xml
localhost:8787
Project
order-edit.xml
workshop (C:\projects\workshop)
modules
core [app-core]
global [app-global]
gui [app-gui]
web [app-web]
.gitignore
build.gradle
settings.gradle
workshop.iml
workshop.ipr
workshop.iws
External Libraries
window dsContext
1: Project
7: Structure
2: Favorites
Maven Projects
Ant Build
Terminal
6: TODO
Event Log
8:16 LF UTF-8

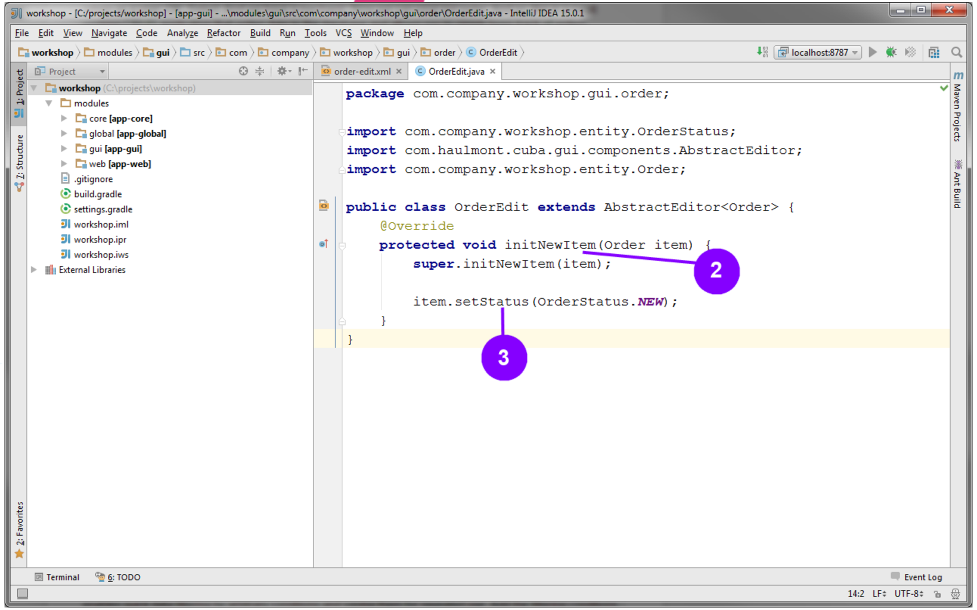
```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<window xmlns="http://schemas.haulmont.com/cuba/window.xsd"
        caption="Order editor"
        class="com.company.workshop.gui.order.OrderEdit"
        datasource="orderDs"
        focusComponent="fieldGroup"
        messagesPack="com.company.workshop.gui.order">
    <dsContext>
        <datasource id="orderDs"
                    class="com.company.workshop.entity.Order"
                    view="order-view">
            <collectionDatasource id="partsDs"
                                  property="parts"/>
        </datasource>
    </dsContext>
    <layout expand="windowActions"
            spacing="true">
        <fieldGroup id="fieldGroup"
                    datasource="orderDs">
            <column width="250px">
                <field id="description"/>
                <field id="hoursSpent"/>
                <field id="amount"/>
                <field id="status"/>
                <field id="client"/>
                <field id="mechanic"/>
            </column>
        </fieldGroup>
        <groupBox id="partsBox"
                  caption="Parts">
            <table id="partsTable"
                   height="200px"
                   width="100%">
                <actions>
                    <action id="add"/>
                    <action id="remove"/>
```

Set default Status for an order

Stay in the IDE and follow the steps:

1. Hold **Ctrl** button and click on ***OrderEdit*** in class attribute of the XML descriptor to navigate to its implementation
2. Override method ***initNewItem***
3. 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);
    }
}
```

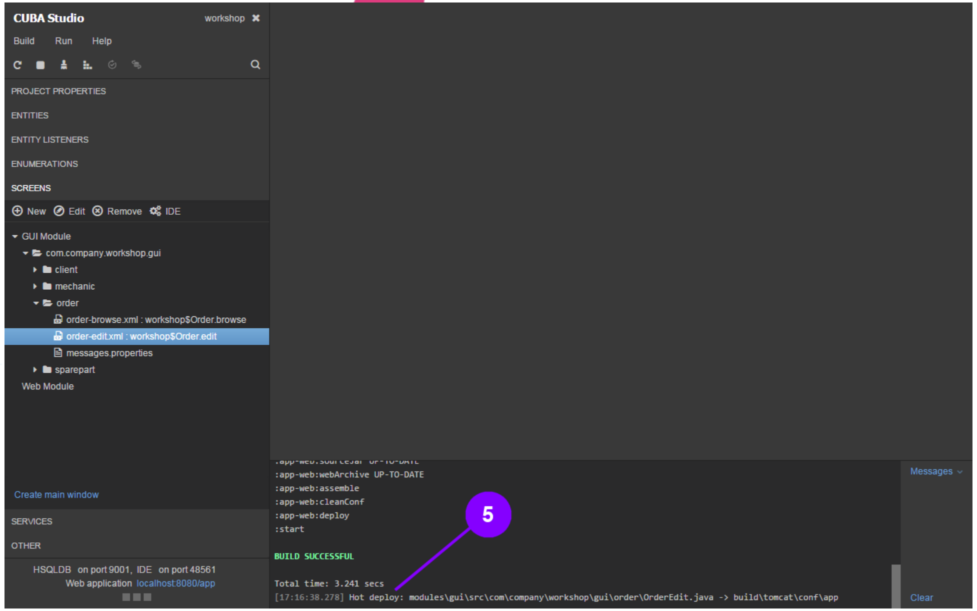

workshop - [C:/projects/workshop] - [app-gui] - ...\modules\gui\src\com\company\workshop\gui\order\OrderEdit.java - IntelliJ IDEA 15.0.1

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

import com.company.workshop.entity.OrderStatus;
import com.haulmont.cuba.gui.components.AbstractEditor;
import com.company.workshop.entity.Order;

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

        item.setStatus(OrderStatus.NEW);
    }
}
```

CUBA.platform
www.cuba-platform.com

Hot deploy

1. Open our application in the browser
2. Open/Reopen **Application — Orders** screen
3. Click **Create**
4. We see our changes, although we haven't restarted the server
5. The CUBA Studio automatically detects and the hot-deploys changes, except for the data model, which saves a lot of time while UI development



```
:app-web:sourceJar UP-TO-DATE
:app-web:webArchive UP-TO-DATE
:app-web:assemble
:app-web:cleanConf
:app-web:deploy
:start

BUILD SUCCESSFUL

Total time: 3.241 secs
[17:16:38.278] Hot deploy: modules\gui\src\com\company\workshop\gui\order\OrderEdit.java -> build\tomcat\conf\app
```

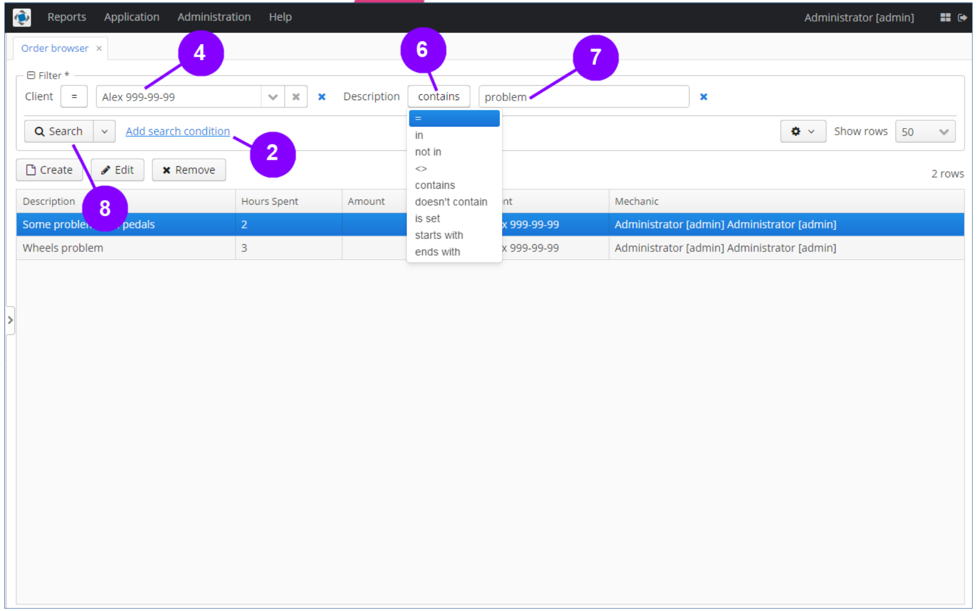
Generic filter

Filter component

1. Add a few orders to the system
2. Click **Add new condition**
3. Select **Client**
4. Set ***Alex*** as value for condition for the **Client** attribute
5. Select **Description**
6. Change **[=]** operation to **[contains]**
7. Enter a word to **Description** field
8. Click **Search**

The filter is a versatile generic tool for filtering lists of entities, typically used on browser screens.

It enables quick data filtering by arbitrary conditions and saving them for repeated use.



Client = Alex 999-99-99 | Description contains problem | Show rows 50

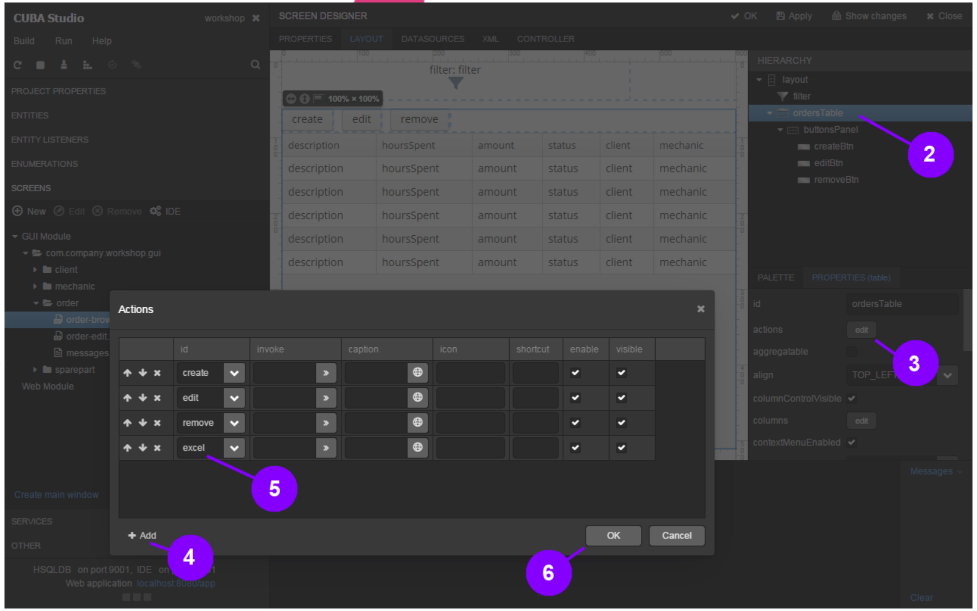
Description	Hours Spent	Amount	nt	Mechanic
Some probl pedals	2		x 999-99-99	Administrator [admin] Administrator [admin]
Wheels problem	3		x 999-99-99	Administrator [admin] Administrator [admin]

Actions

Standard actions

The standard screens contain **Create**, **Edit**, and **Remove** actions by default.
Let’s add an action to **export** the order list to **Excel**.

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**



id	ordersTable
actions	edit
aggregatable	
align	TOP_LEFT
columnControlVisible	✓
columns	edit
contextMenuEnabled	✓

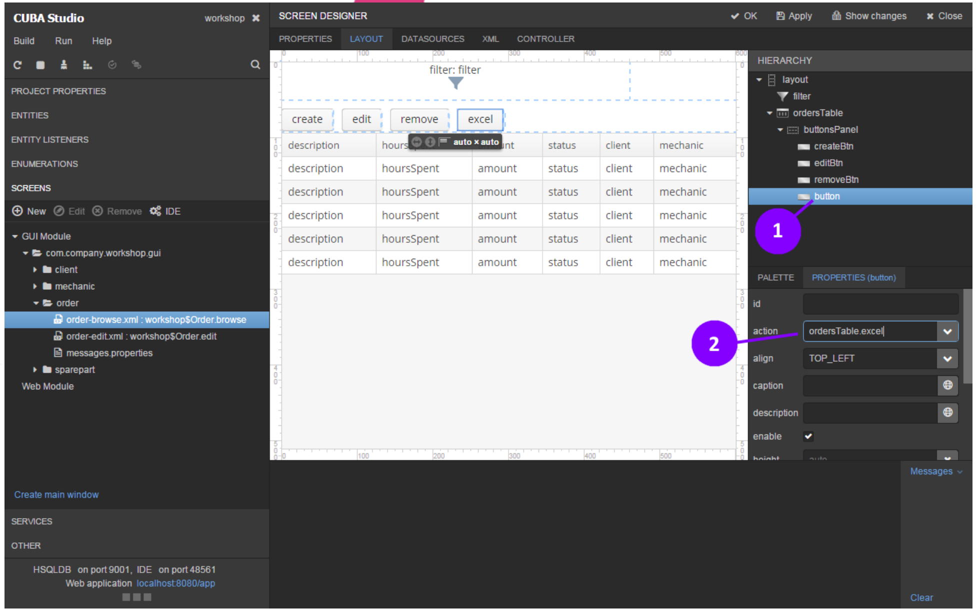
id	invoke	caption	icon	shortcut	enable	visible
create					✓	✓
edit					✓	✓
remove					✓	✓
excel					✓	✓

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

Excel action

1. Add a new button to the button panel (**drug and drop** it into the hierarchy of components)
2. Select ***ordersTable.excel*** action for button using properties panel
3. Save the screen
4. Open/Reopen the **Orders** screen
5. Click **Excel** to export your orders to an xls file

The platform has standard actions for common operations: **Create**, **Edit**, **Remove**, **Include**, **Exclude** (for sets), **Refresh**, **Excel**, and you can create your own actions.



description	hoursSpent	amount	status	client	mechanic
description	hoursSpent	amount	status	client	mechanic
description	hoursSpent	amount	status	client	mechanic
description	hoursSpent	amount	status	client	mechanic
description	hoursSpent	amount	status	client	mechanic
description	hoursSpent	amount	status	client	mechanic

id	
action	ordersTable.excel
align	TOP_LEFT
caption	
description	
enable	✓
height	auto

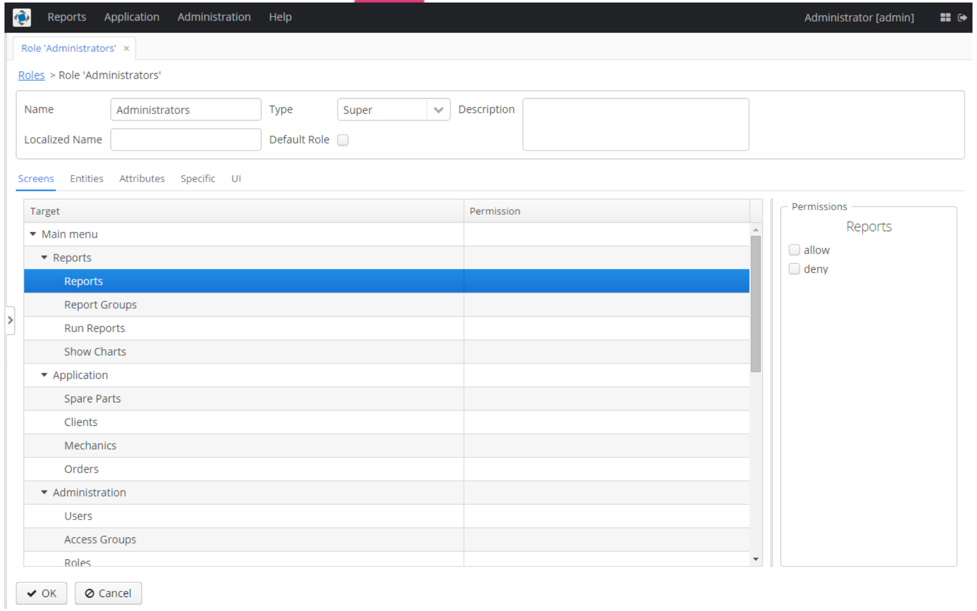
Security

Security subsystem

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.



Name	Administrators
Type	Super
Description	
Localized Name	
Default Role	☐

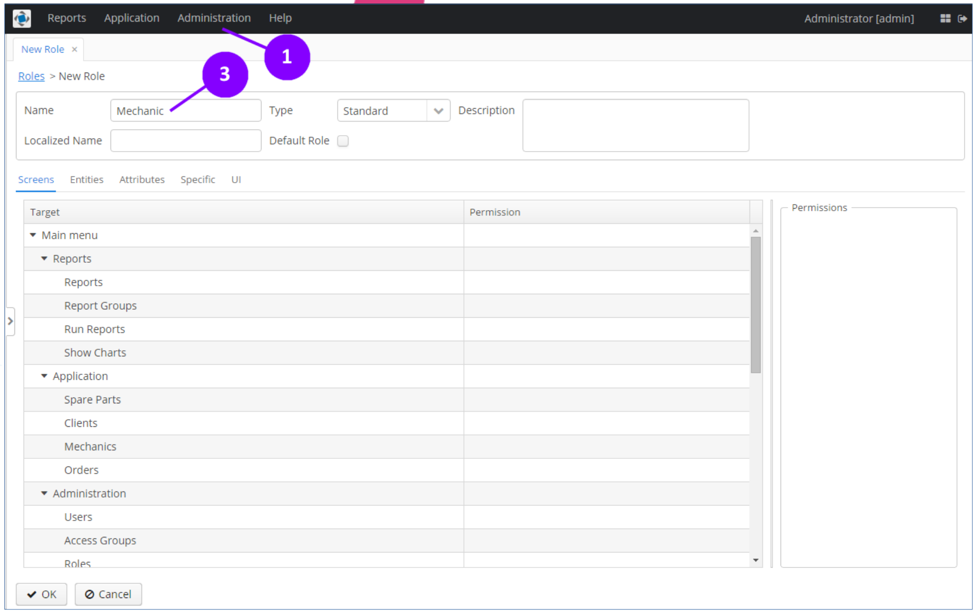
Target	Permission
Main menu	
Reports	
Reports	
Report Groups	
Run Reports	
Show Charts	
Application	
Spare Parts	
Clients	
Mechanics	
Orders	
Administration	
Users	
Access Groups	
Roles	

Permissions
Reports
☐ allow
☐ deny

Mechanic role

We need the **Mechanic role** for 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***



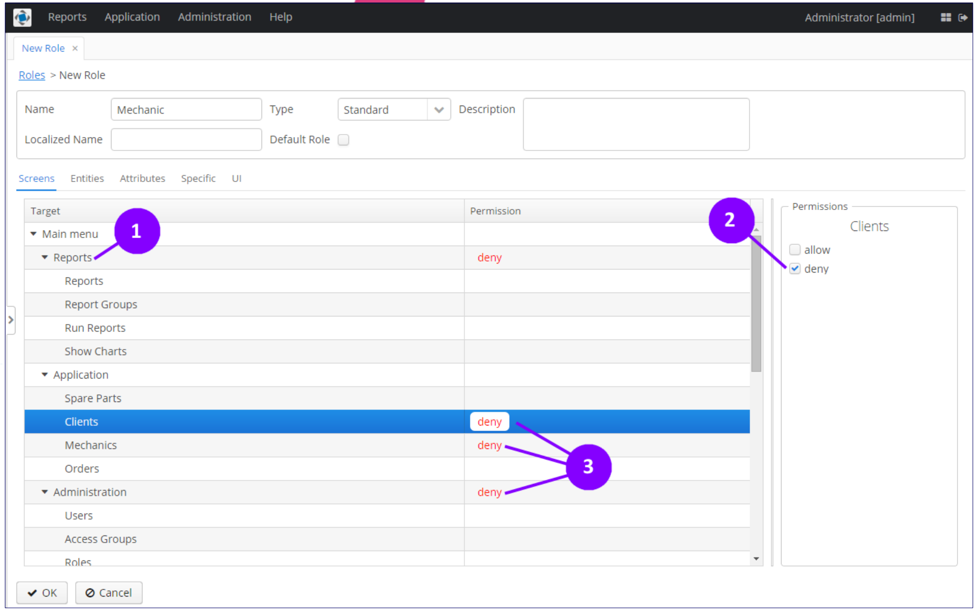
Name	Mechanic	Type	Standard	Description	
Localized Name		Default Role	☐		

Target	Permission
Main menu	
Reports	
Reports	
Report Groups	
Run Reports	
Show Charts	
Application	
Spare Parts	
Clients	
Mechanics	
Orders	
Administration	
Users	
Access Groups	
Roles	

Screen permissions

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

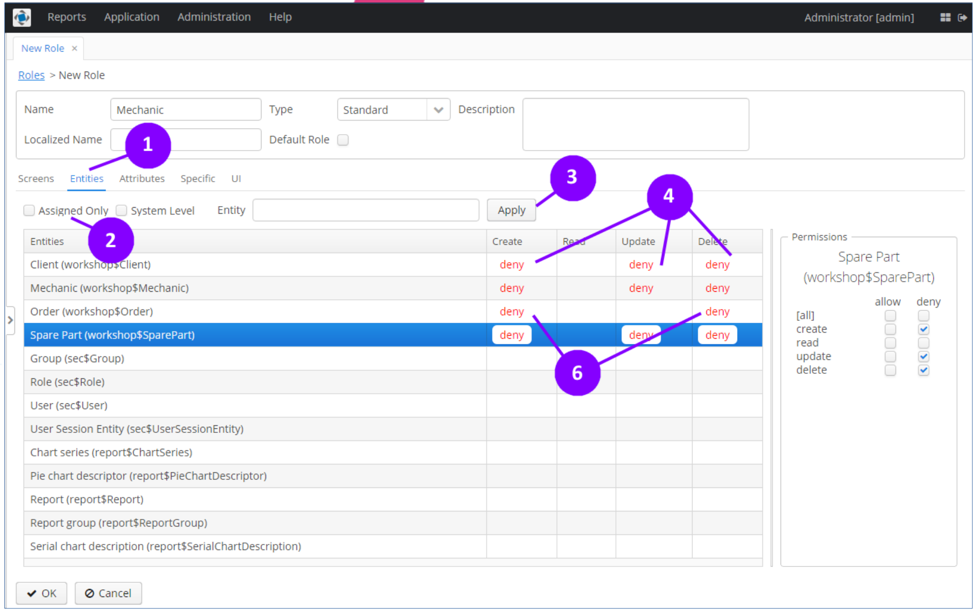
1. Select **Reports** row in the table with **Screens**
2. Select **deny** checkbox at the right
3. Similarly deny access for **Administration**, **Clients** and **Mechanics**



Target	Permission
Main menu	
Reports	deny
Reports	
Report Groups	
Run Reports	
Show Charts	
Application	
Spare Parts	
Clients	deny
Mechanics	deny
Orders	
Administration	deny
Users	
Access Groups	
Roles	

CRUD permissions

1. Open the **Entities** tab
2. Unset the **Assigned Only** checkbox
3. Click **Apply**
4. Select the ***Client*** entity and forbid **create**, **update** and **delete** operations
5. Same for the ***Mechanic*** and ***SparePart*** entities
6. For **Order**, we'll restrict only **create** and **delete**



Name	Mechanic	Type	Standard	Description	
Localized Name		Default Role	☐		

Screens | **Entities** | Attributes | Specific | UI

☐ Assigned Only ☐ System Level Entity [ ] Apply

Entities	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$User)				
User Session Entity (sec$UserSessionEntity)				
Chart series (report$ChartSeries)				
Pie chart descriptor (report$PieChartDescriptor)				
Report (report$Report)				
Report group (report$ReportGroup)				
Serial chart description (report$SerialChartDescription)				

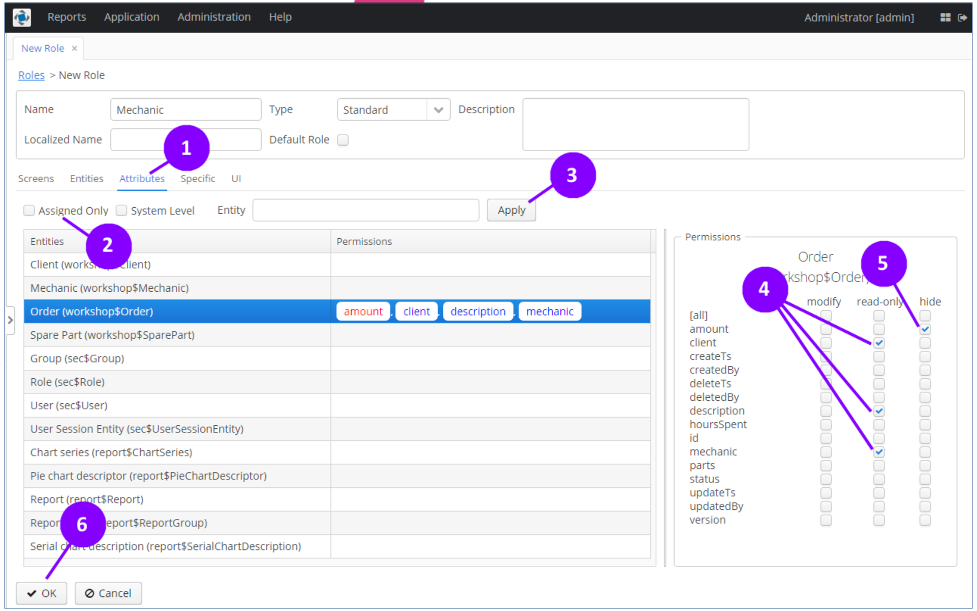
Permissions

Spare Part (workshop$SparePart)

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

Attribute permissions

1. Open the **Attributes** tab
2. Unset the **Assigned Only** checkbox
3. Click **Apply**
4. Select **Order** row and tick **read only** for ***client***, ***mechanic*** and ***description***
5. Set **hide** for amount attribute
6. Click **OK** to save the role



Roles > New Role

Name	Mechanic	Type	Standard	Description	
Localized Name		Default Role	☐		

Screens Entities Attributes Specific UI

☐ Assigned Only ☐ System Level Entity

Entities	Permissions
Client (workshop$Client)	
Mechanic (workshop$Mechanic)	
Order (workshop$Order)	amount, client, description, mechanic
Spare Part (workshop$SparePart)	
Group (sec$Group)	
Role (sec$Role)	
User (sec$User)	
User Session Entity (sec$UserSessionEntity)	
Chart series (report$ChartSeries)	
Pie chart descriptor (report$PieChartDescriptor)	
Report (report$Report)	
Report (report$ReportGroup)	
Serial chart description (report$SerialChartDescription)	

Permissions

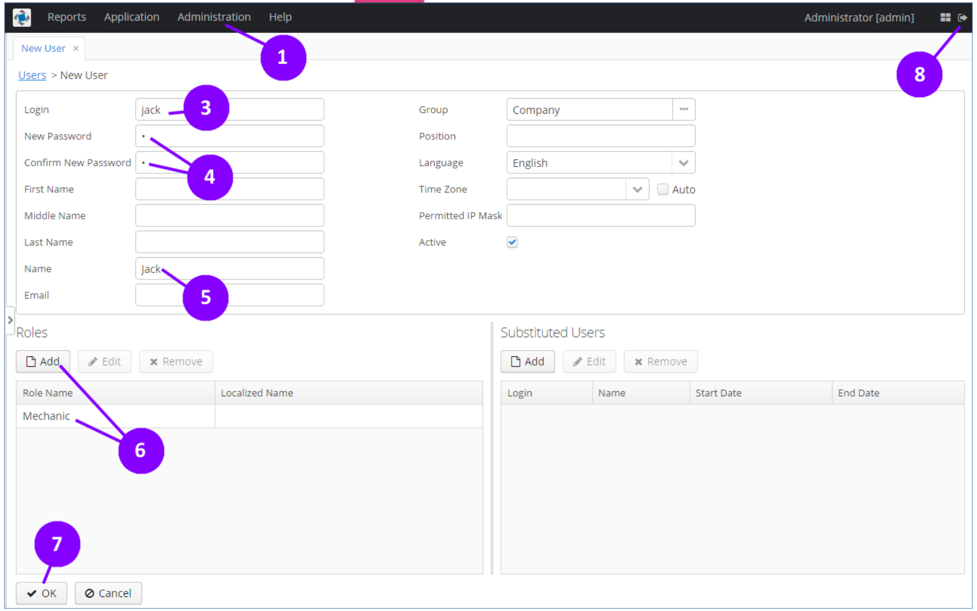
Order
workshop$Order

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

CUBA.platform
www.cuba-platform.com

New user

1. Open **Administration — Users** from the menu
2. Click **Create**
3. Set **Login**: ***jack***
4. Specify password and password confirmation
5. Set **Name**: **Jack**
6. Add the ***Mechanic*** role to user ***Roles***
7. Click **OK** to save the user
8. Click on **exit** icon at the top right corner of application window



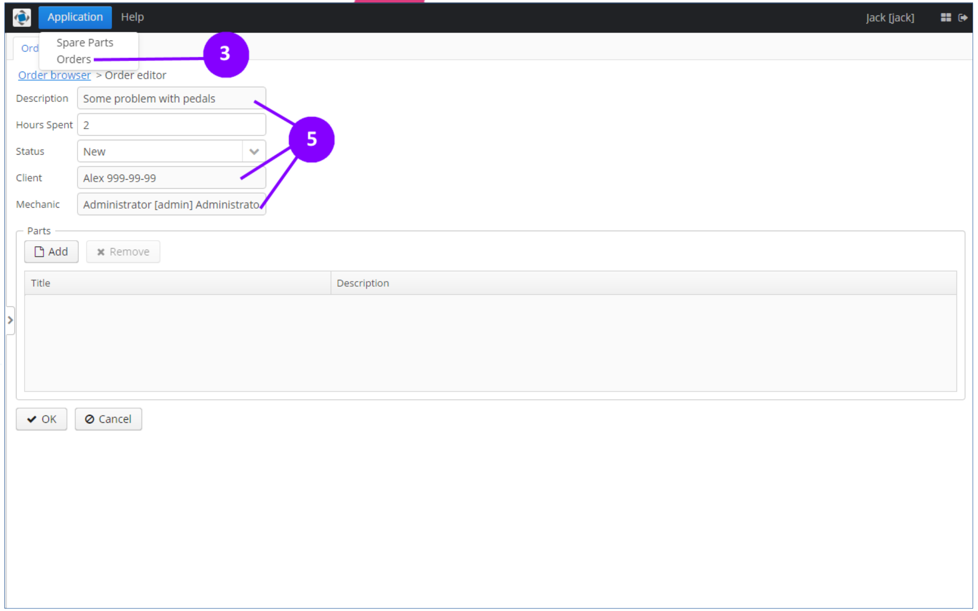
Login	jack
New Password	•
Confirm New Password	•
First Name	
Middle Name	
Last Name	
Name	Jack
Email	
Group	Company
Position	
Language	English
Time Zone	☐ Auto
Permitted IP Mask	
Active	✓

Role Name	Localized Name
Mechanic	

Login	Name	Start Date	End Date

Role-based security in action

1. Login to the system as ***jack***
2. **Reports** and **Administrations** menus are now hidden
3. Open **Application — Orders** from the menu
4. **Edit** existing order
5. The ***description***, ***client*** and ***mechanic*** fields are readonly
6. The ***amount*** field is hidden



Order browser > Order editor

Description	Some problem with pedals
Hours Spent	2
Status	New
Client	Alex 999-99-99
Mechanic	Administrator [admin] Administrato

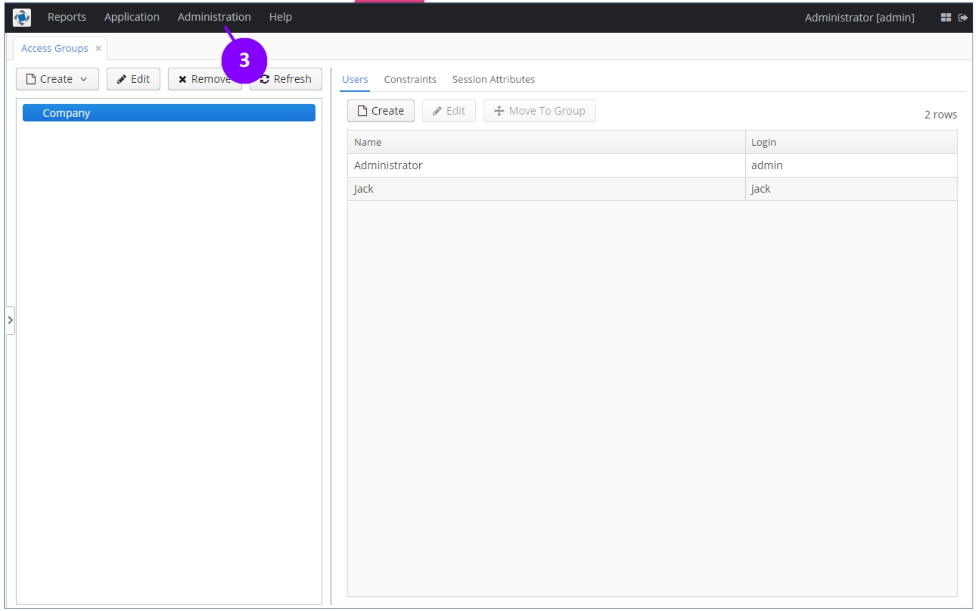
Title	Description

Row level security

What about the visibility of orders for the mechanic?
Let's limit the list of displayed orders to the logged in mechanic's orders only. We will use the **access group** mechanism for this.

1. **Log out** from the system
2. Log in as ***admin***
3. Open **Administration — Access Groups** from the menu

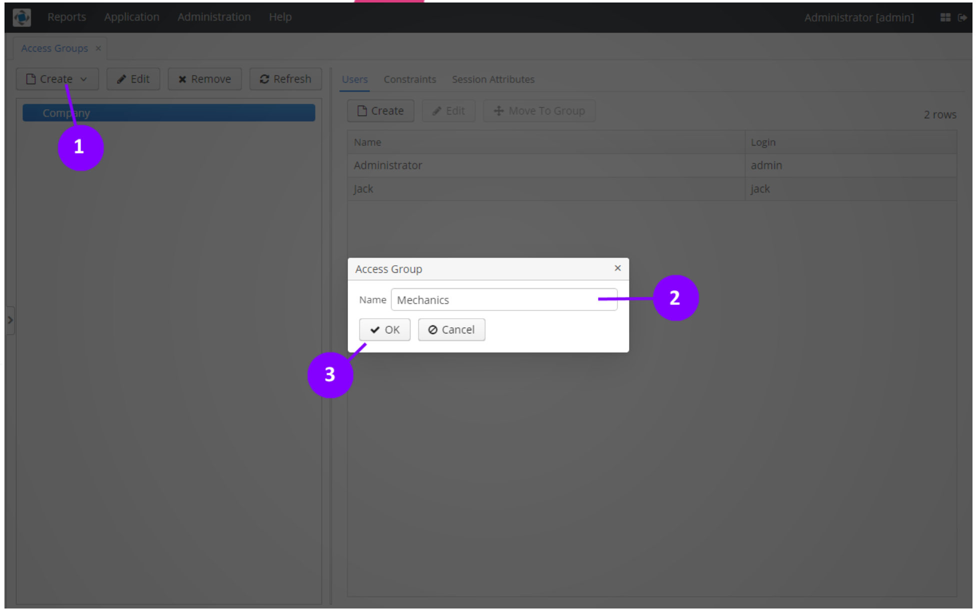
The groups have hierarchical structure, where each element defines a set of constraints, allowing controlling access to individual entity instances (at row level).



Name	Login
Administrator	admin
Jack	jack

Create an access group

1. Click **Create — New**
2. Set **Name**: ***Mechanics***
3. Click **OK**



Name	Login
Administrator	admin
Jack	jack

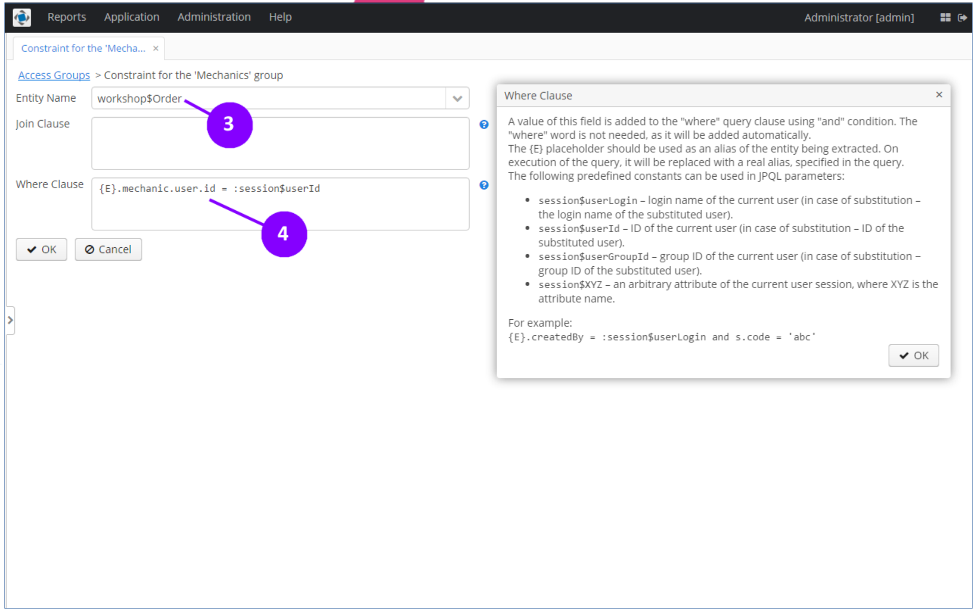
Name: Mechanics

Add constraint for the access group

1. Open the **Constraints** tab for the newly created group
2. Click **Create** in the **Constraints** tab
3. Select Entity Name: ***workshop$Order***
4. Enter condition to **Where Clause** as following

{E}.mechanic.user.id = :session$userId

,where ***{E}*** is a generic alias for the entity


Reports
Application
Administration
Help
Administrator [admin]
Constraint for the 'Mecha...
Access Groups > Constraint for the 'Mechanics' group
3
4
OK
Cancel

Entity Name: workshop$Order
Join Clause:
Where Clause: {E}.mechanic.user.id = :session$userId

Where Clause

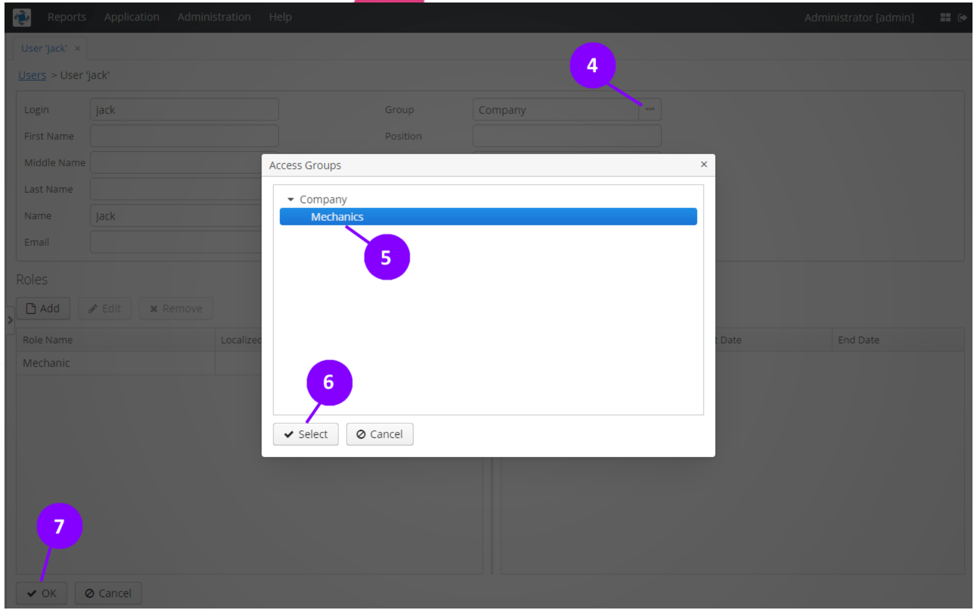
A value of this field is added to the "where" query clause using "and" condition. The "where" word is not needed, as it will be added automatically.
The {E} placeholder should be used as an alias of the entity being extracted. On execution of the query, it will be replaced with a real alias, specified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

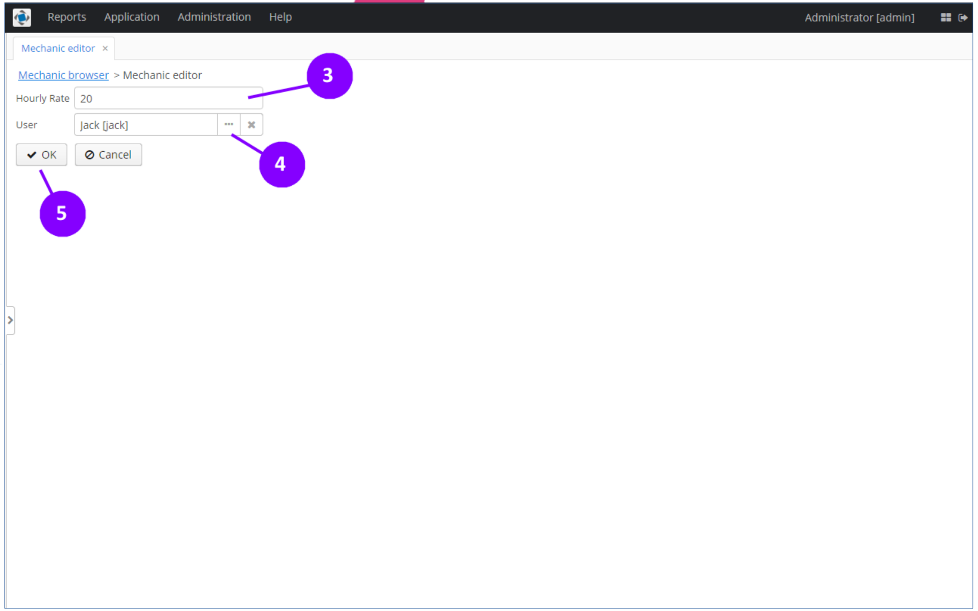
Assign group to the user

1. Click **OK** to save the constraint
2. Open **Administration — Users** from the menu
3. Edit the user with login: ***jack***
4. Click on button **[…]** at the right of the Group field
5. Select the **Mechanics** group
6. Click **Select**
7. Click **OK** to save the user



Create a mechanic for the user

1. Open **Application — Mechanic** from the menu
2. Click **Create**
3. Set **Hourly Rate**
4. Select user: ***jack***
5. Click **OK** to save the mechanic

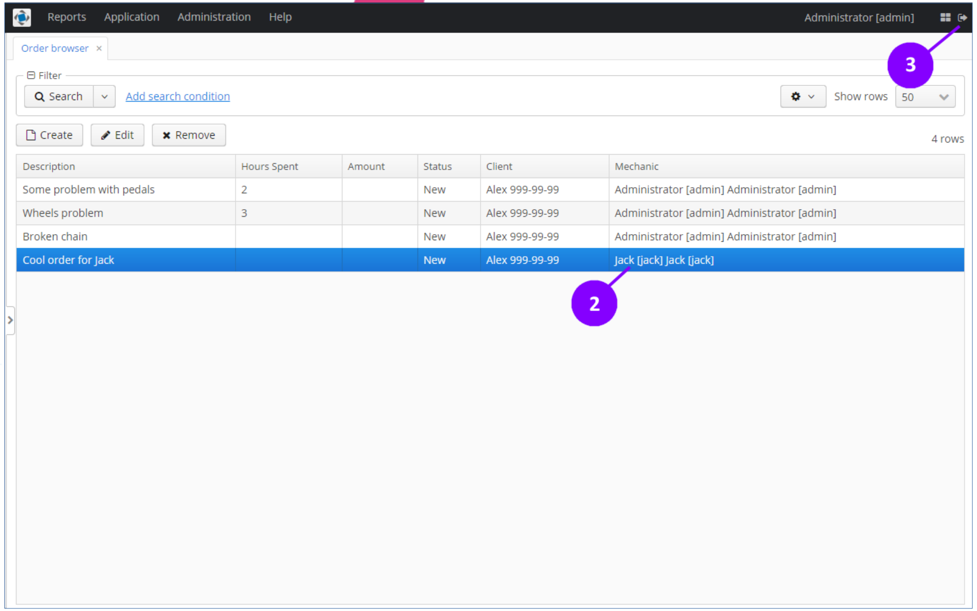


Mechanic browser > Mechanic editor

Hourly Rate	20
User	jack [jack]

Create an order for the mechanic

1. Open **Application — Orders** from the menu
2. Create order for **Jack**
3. **Log out** from the system

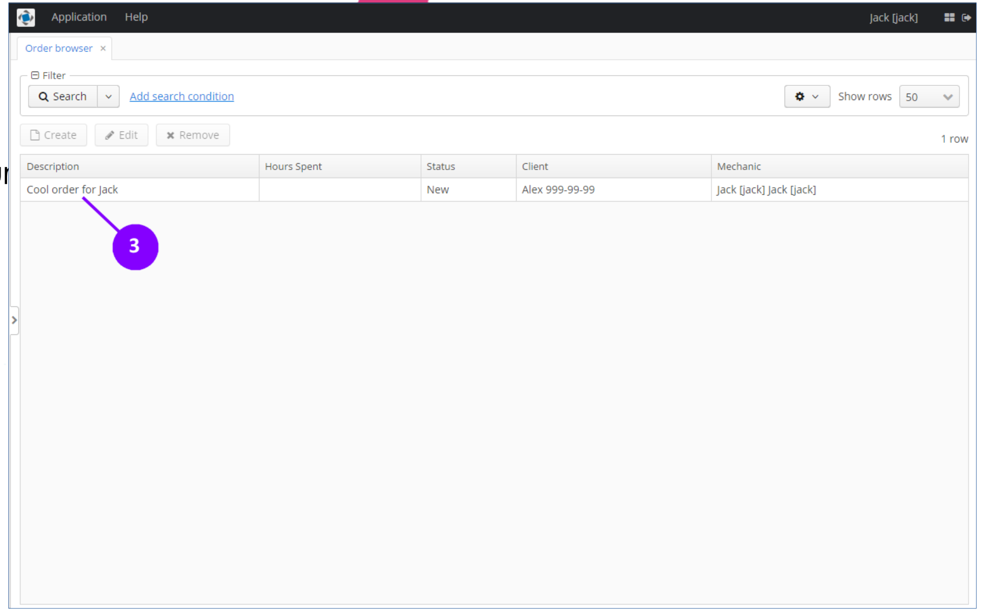


Description	Hours Spent	Amount	Status	Client	Mechanic
Some problem with pedals	2		New	Alex 999-99-99	Administrator [admin] Administrator [admin]
Wheels problem	3		New	Alex 999-99-99	Administrator [admin] Administrator [admin]
Broken chain			New	Alex 999-99-99	Administrator [admin] Administrator [admin]
Cool order for Jack			New	Alex 999-99-99	Jack [jack] Jack [jack]

Row level security in action

1. Log in to the system as ***jack***
2. Open **Application — Orders** from the menu
3. We see only one order for Jack!

We have restricted access for particular orders only to the mechanics who perform them. The access groups functionality allows you to configur the Row-level security in your application completely transparent for your application code without interfering with a screen code.



Description	Hours Spent	Status	Client	Mechanic
Cool order for Jack		New	Alex 999-99-99	Jack [jack] Jack [jack]

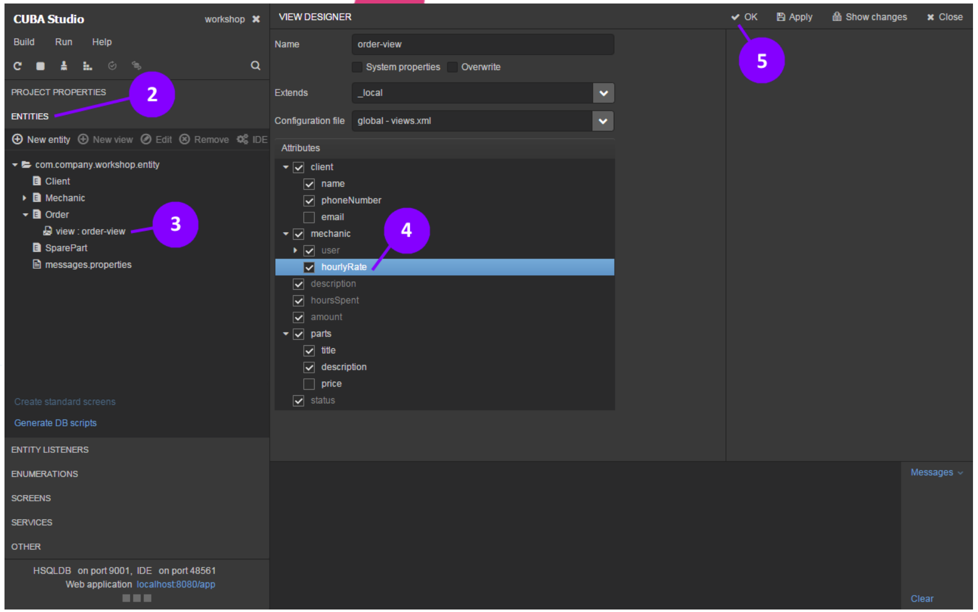
Services

Services

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.

To use mechanic hourly rate, we'll need to load this attribute, so we need to add it to **order-view**.

1. Switch to the Studio
2. Open the **Entities** section of the Studio navigation panel
3. Edit ***order-view***
4. Include the **hourlyRate** attribute to the view
5. Click **OK** to save the view



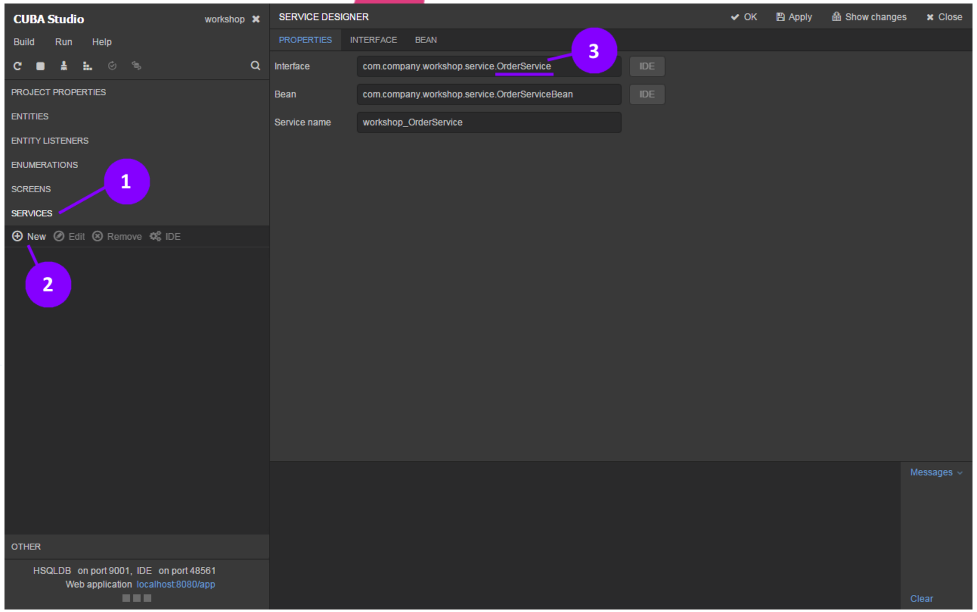
Name: order-view
☐ System properties ☐ Overwrite
Extends: _local
Configuration file: global - views.xml

Attributes
- ✓ client
  - ✓ name
  - ✓ phoneNumber
  - ☐ email
- ✓ mechanic
  - ✓ user
  - ✓ hourlyRate
- ✓ description
- ✓ hoursSpent
- ✓ amount
- ✓ parts
  - ✓ title
  - ✓ description
  - ☐ price
- ✓ status

Generate Service stub

Business logic changes can happen very often, so it would be better to put it in a separate class - a service that different system parts will be able to invoke to calculate the price for repair. Let's create a stub for such service from the Studio and implement the price calculation logic there. And in our screen, we'll create the method to invoke this service.

1. Go to the **Services** section in the Studio
2. Click **New**
3. Change the last part of Interface name to ***OrderService***



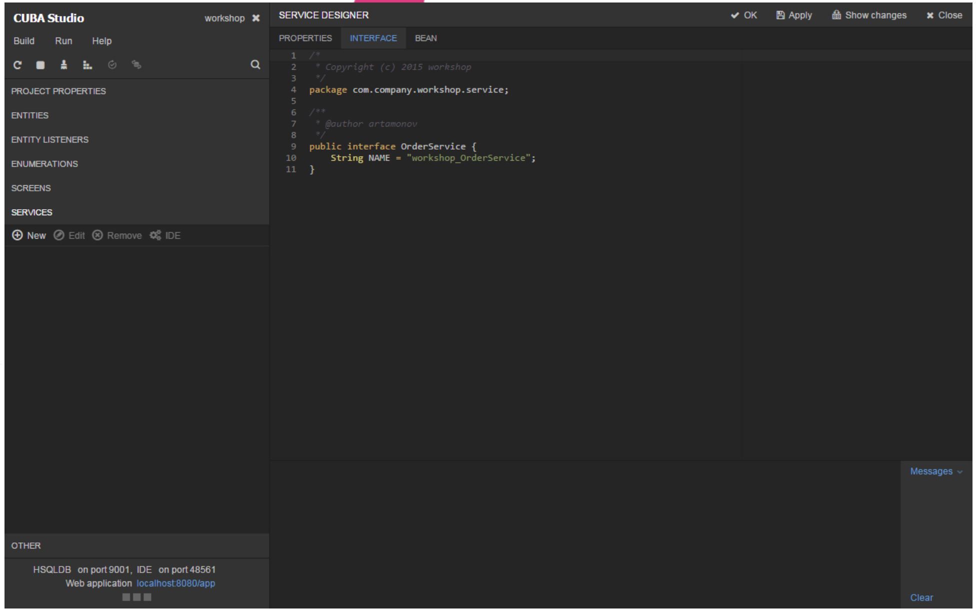
Interface: com.company.workshop.service.OrderService
Bean: com.company.workshop.service.OrderServiceBean
Service name: workshop_OrderService

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

Service interface and bean generation

In the **Interface** tab we can see the source code of the service interface, the **Bean** tab shows its implementation. 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).



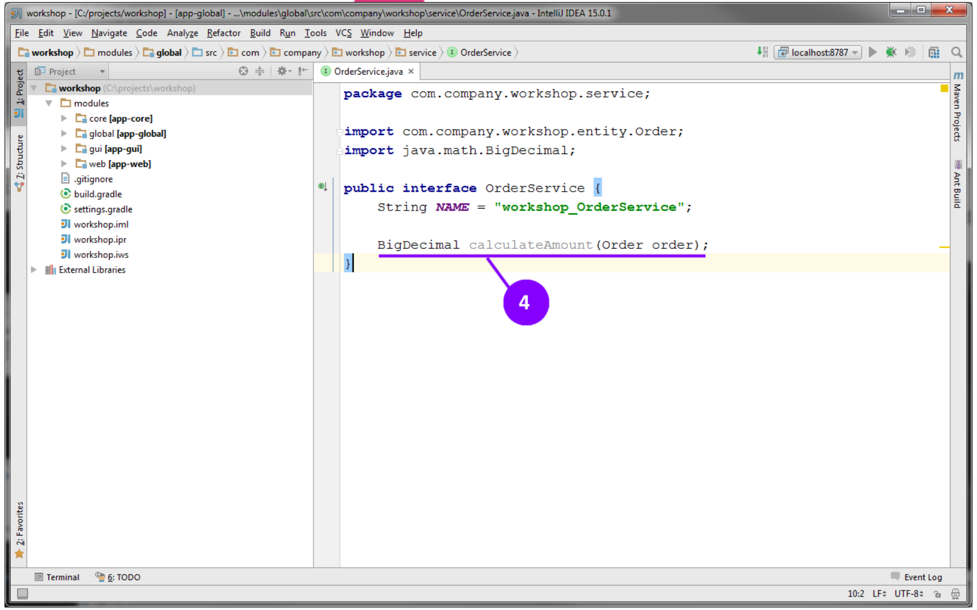
```
/*
 * Copyright (c) 2015 workshop
 */
package com.company.workshop.service;

/**
 * @author artamonov
 */
public interface OrderService {
    String NAME = "workshop_OrderService";
}
```

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

Add method to a service

1. Click **OK** to save interface stub
2. Select the ***OrderService*** item in the navigation panel
3. Click **IDE**
4. 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);
}
```



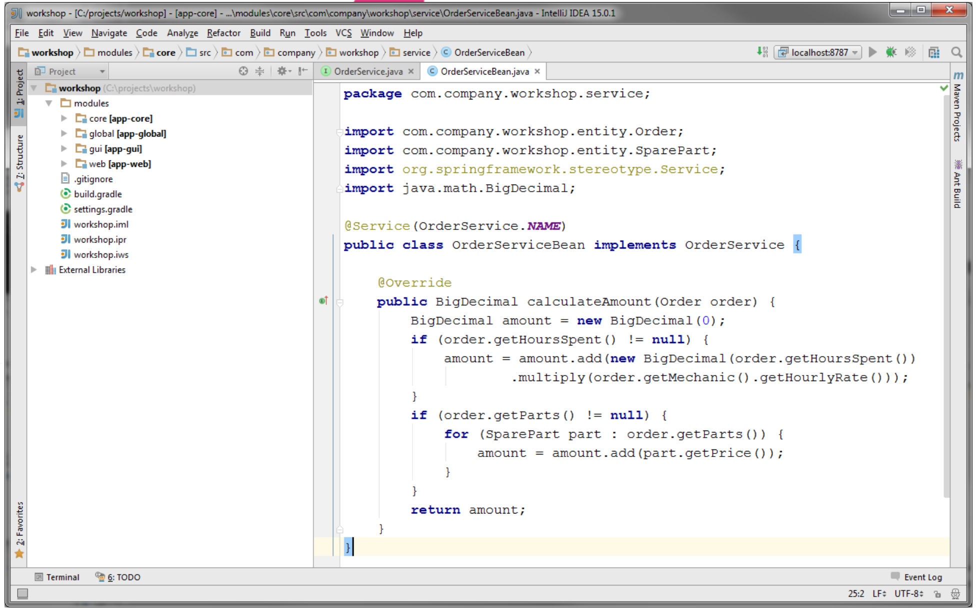
workshop - [C:/projects/workshop] - [app-global] - ...\modules\global\src\com\company\workshop\service\OrderService.java - IntelliJ IDEA 15.0.1

```
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);
}
```

Service method implementation

1. Go to ***OrderServiceBean*** using the green navigation icon at the left
2. Implement the method

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

import com.company.workshop.entity.*;
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;
    }
}
```

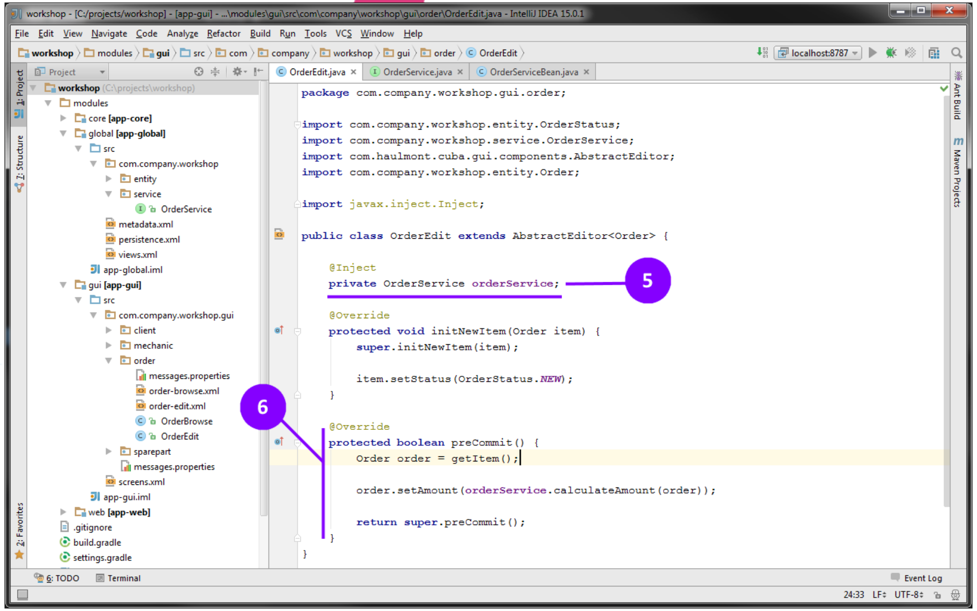

workshop - [C:/projects/workshop] - [app-core] - ...\modules\core\src\com\company\workshop\service\OrderServiceBean.java - IntelliJ IDEA 15.0.1
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
workshop › modules › core › src › com › company › workshop › service › OrderServiceBean
localhost:8787
Project
workshop (C:\projects\workshop)
modules
core [app-core]
global [app-global]
gui [app-gui]
web [app-web]
.gitignore
build.gradle
settings.gradle
workshop.iml
workshop.ipr
workshop.iws
External Libraries
OrderService.java
OrderServiceBean.java
1: Project
7: Structure
2: Favorites
Maven Projects
Ant Build
Terminal
6: TODO
Event Log
25:2 LF: UTF-8

```java
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;
    }
}
```

Call the service method from UI

1. Go back to the Studio
2. Select the **order-edit.xml** screen in the **Screens** section of the navigation panel
3. Click **IDE**
4. Go to the screen controller (***OrderEdit*** class)
5. Add ***OrderService*** field to class and annotate it with **@Inject** annotation
6. Override the ***preCommit()*** method and invoke the calculation method of ***OrderService***

```
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();
    }
}
```


workshop - [C:/projects/workshop] - [app-gui] - ...\modules\gui\src\com\company\workshop\gui\order\OrderEdit.java - IntelliJ IDEA 15.0.1
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
workshop > modules > gui > src > com > company > workshop > gui > order > OrderEdit
localhost:8787
Project
OrderEdit.java × OrderService.java × OrderServiceBean.java ×
1: Project
7: Structure
2: Favorites
Ant Build
Maven Projects
6: TODO
Terminal
Event Log
24:33 LF UTF-8
5
6

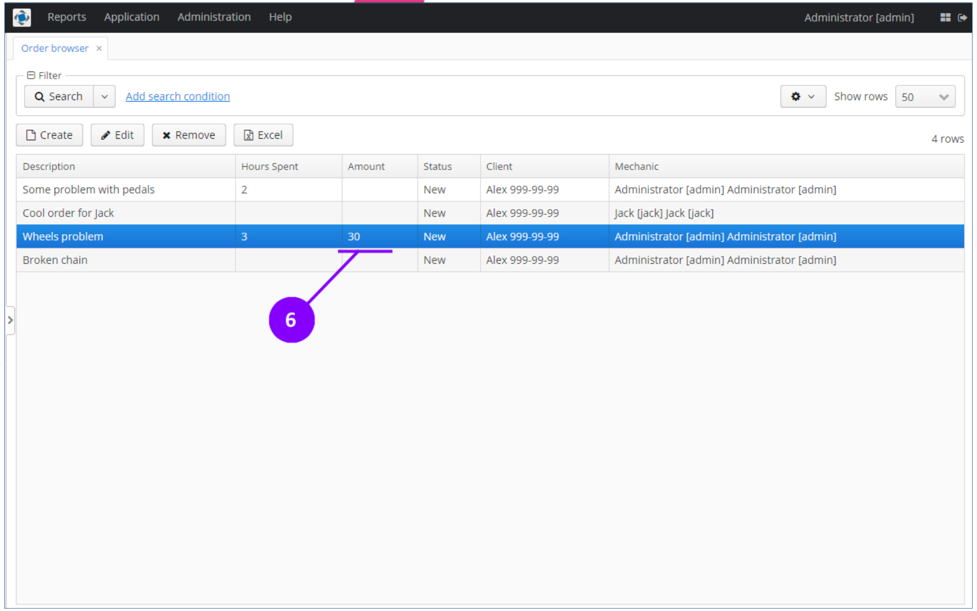
```
workshop (C:\projects\workshop)
  modules
    core [app-core]
    global [app-global]
      src
        com.company.workshop
          entity
          service
            OrderService
          metadata.xml
          persistence.xml
          views.xml
      app-global.iml
    gui [app-gui]
      src
        com.company.workshop.gui
          client
          mechanic
          order
            messages.properties
            order-browse.xml
            order-edit.xml
            OrderBrowse
            OrderEdit
          sparepart
          messages.properties
        screens.xml
      app-gui.iml
    web [app-web]
  .gitignore
  build.gradle
  settings.gradle
```

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

import com.company.workshop.entity.OrderStatus;
import com.company.workshop.service.OrderService;
import com.haulmont.cuba.gui.components.AbstractEditor;
import com.company.workshop.entity.Order;

import javax.inject.Inject;

public class OrderEdit extends AbstractEditor<Order> {

    @Inject
    private OrderService orderService;

    @Override
    protected void initNewItem(Order item) {
        super.initNewItem(item);

        item.setStatus(OrderStatus.NEW);
    }

    @Override
    protected boolean preCommit() {
        Order order = getItem();

        order.setAmount(orderService.calculateAmount(order));

        return super.preCommit();
    }
}
```

CUBA.platform
www.cuba-platform.com

Test the service call

1. Restart your application using the **Run — Restart application** action from the Studio
2. Open **Application — Orders** from the menu
3. Open **editor screen** for any order
4. Set **Hours Spent**
5. Click **OK** to save order
6. We can see a newly calculated value of the amount in the table



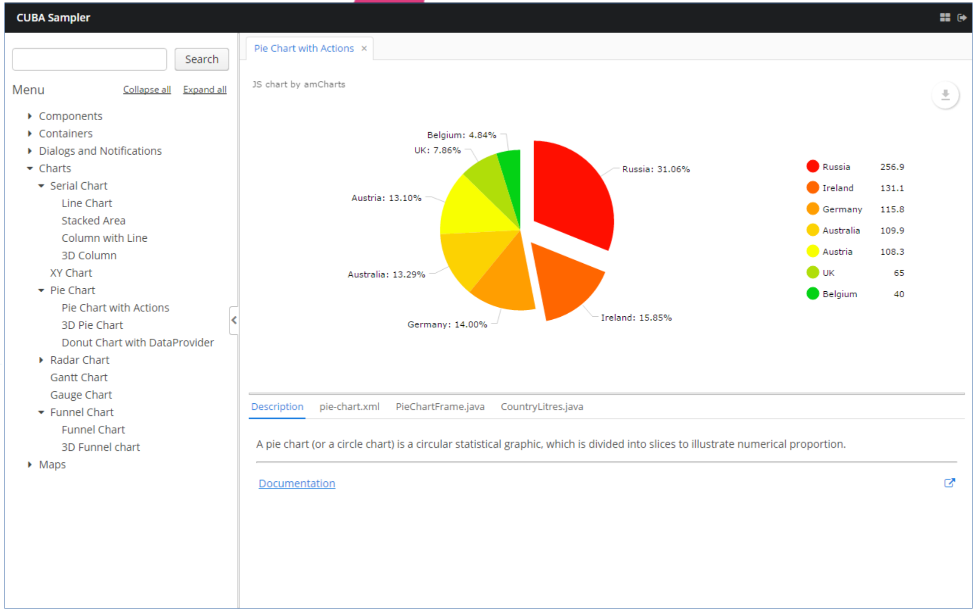
Description	Hours Spent	Amount	Status	Client	Mechanic
Some problem with pedals	2		New	Alex 999-99-99	Administrator [admin] Administrator [admin]
Cool order for Jack			New	Alex 999-99-99	Jack [jack] Jack [jack]
Wheels problem	3	30	New	Alex 999-99-99	Administrator [admin] Administrator [admin]
Broken chain			New	Alex 999-99-99	Administrator [admin] Administrator [admin]

Charts

Charts

Let’s assume our mechanic uses and likes the application but now he wants to add statistics. He wants a chart showing the amount of orders per mechanic to reward them at the end of the month.

To implement this functionality we’ll use the **charts** module of the CUBA platform, based on AmCharts. It allows us to display interactive charts in a web application based on system data and specify chart configuration via XML.



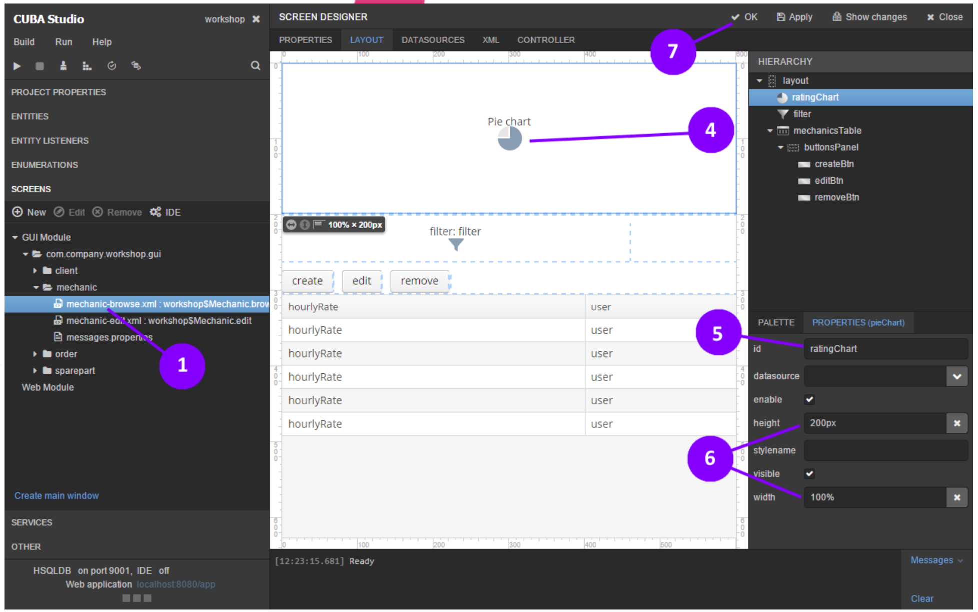
●	Russia	256.9
●	Ireland	131.1
●	Germany	115.8
●	Australia	109.9
●	Austria	108.3
●	UK	65
●	Belgium	40

A pie chart (or a circle chart) is a circular statistical graphic, which is divided into slices to illustrate numerical proportion.

Add chart component to screen

Let's place the work distribution chart on the mechanics browser screen.

1. Open **mechanic-browse.xml** screen in the Studio
2. Place the cursor into the components palette, type ***Chart***
3. The Studio will filter the component list and show us components to display charts
4. Drag **PieChart** and drop it to the UI editor area
5. Set id for chart: ***ratingChart***
6. Set width 100% and height 200px using **Properties** panel
7. Click **OK** to save the screen



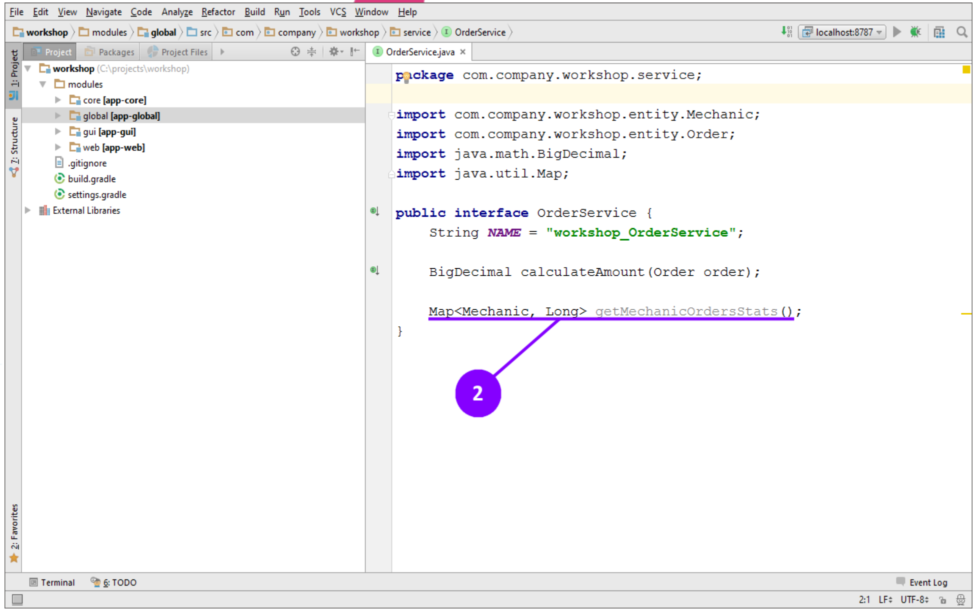
id	ratingChart
datasource	
enable	✓
height	200px
stylename	
visible	✓
width	100%

Load data for chart

To load data for our chart, let's declare a new method in ***OrderService***.

1. Go to ***OrderService*** from the Studio by selecting the service and clicking the **IDE** button
2. Add the method definition to the interface:

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

import com.company.workshop.entity.Mechanic;
import com.company.workshop.entity.Order;
import java.math.BigDecimal;
import java.util.Map;

public interface OrderService {
    String NAME = "workshop_OrderService";

    BigDecimal calculateAmount(Order order);

    Map<Mechanic, Long> getMechanicOrdersStats();
}
```



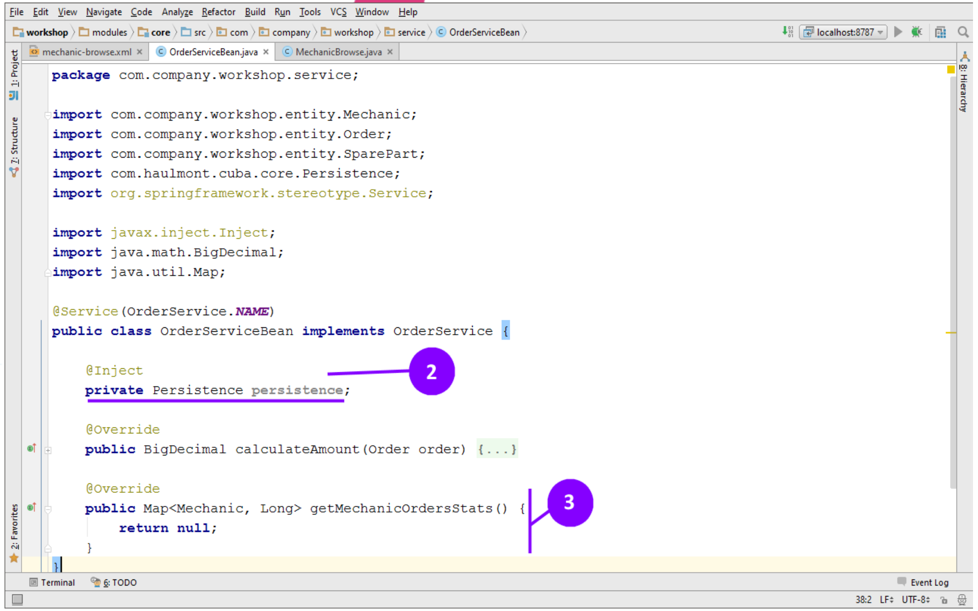
```
package com.company.workshop.service;

import com.company.workshop.entity.Mechanic;
import com.company.workshop.entity.Order;
import java.math.BigDecimal;
import java.util.Map;

public interface OrderService {
    String NAME = "workshop_OrderService";

    BigDecimal calculateAmount(Order order);

    Map<Mechanic, Long> getMechanicOrdersStats();
}
```

CUBA Persistence

The method will retrieve the number of orders for each mechanic from the database using a JPQL query.

Persistence interface is responsible for interaction with the database and allows you to create transactions and execute operations using ***EntityManager***.

1. Open the ***OrderServiceBean*** class
2. Inject the ***Persistence*** object into the class
3. Add stub for ***getMechanicsOrdersStats*** method


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
workshop › modules › core › src › com › company › workshop › service › OrderServiceBean
localhost:8787
mechanic-browse.xml
OrderServiceBean.java
MechanicBrowse.java
1: Project
Z: Structure
2: Favorites
8: Hierarchy
2
3
Terminal
6: TODO
Event Log
38:2 LF: UTF-8:

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

import com.company.workshop.entity.Mechanic;
import com.company.workshop.entity.Order;
import com.company.workshop.entity.SparePart;
import com.haulmont.cuba.core.Persistence;
import org.springframework.stereotype.Service;

import javax.inject.Inject;
import java.math.BigDecimal;
import java.util.Map;

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

    @Inject
    private Persistence persistence;

    @Override
    public BigDecimal calculateAmount(Order order) {...}

    @Override
    public Map<Mechanic, Long> getMechanicOrdersStats() {
        return null;
    }
}
```

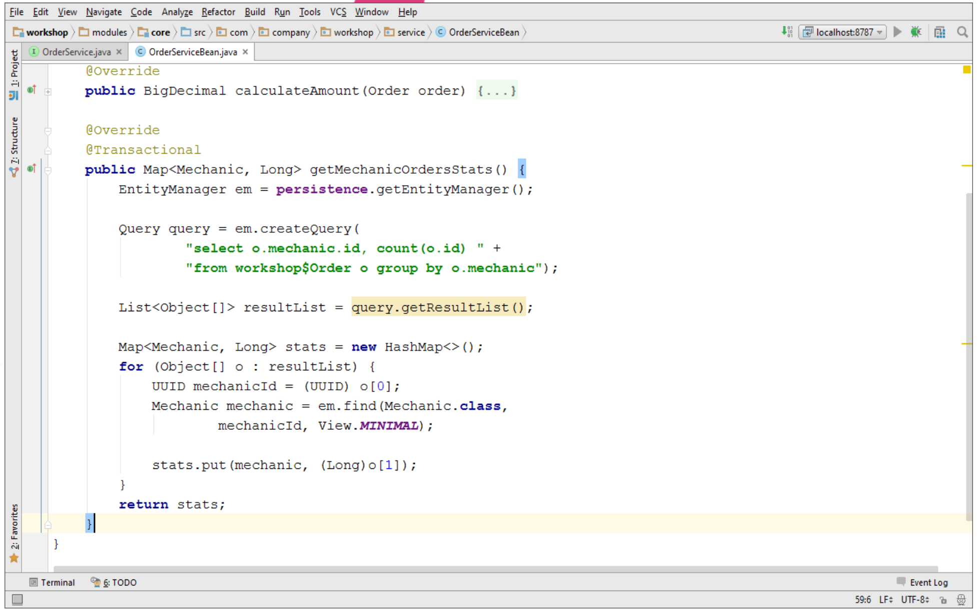
JPQL Query

We'll use the following trivial JPQL query to get the number of orders for each mechanic:

select o.mechanic, count(o.id) from workshop$Order o group by o.mechanic

It aggregates orders by the ***mechanic*** field and returns the number of orders for each mechanic.

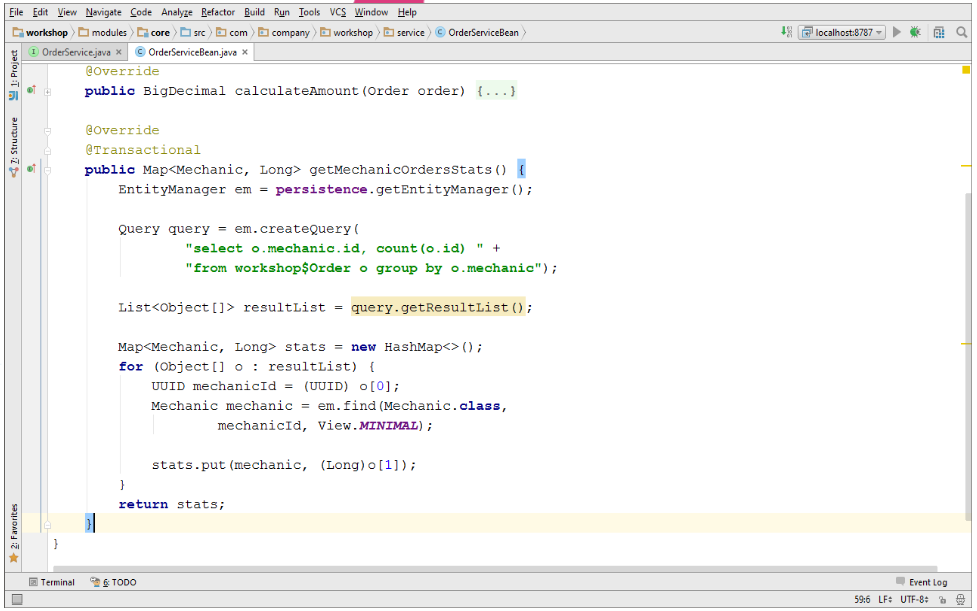
The complete implementation of the method is available on the next slide.


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
workshop › modules › core › src › com › company › workshop › service › OrderServiceBean
localhost:8787
OrderService.java × OrderServiceBean.java ×
1: Project
Z: Structure
2: Favorites
Terminal
6: TODO
Event Log
59:6 LF: UTF-8:

```
    @Override
    public BigDecimal calculateAmount(Order order) {...}

    @Override
    @Transactional
    public Map<Mechanic, Long> getMechanicOrdersStats() {
        EntityManager em = persistence.getEntityManager();

        Query query = em.createQuery(
                "select o.mechanic.id, count(o.id) " +
                "from workshop$Order o group by o.mechanic");

        List<Object[]> resultList = query.getResultList();

        Map<Mechanic, Long> stats = new HashMap<>();
        for (Object[] o : resultList) {
            UUID mechanicId = (UUID) o[0];
            Mechanic mechanic = em.find(Mechanic.class,
                    mechanicId, View.MINIMAL);

            stats.put(mechanic, (Long)o[1]);
        }
        return stats;
    }
}
```

Data loading using CUBA Persistence

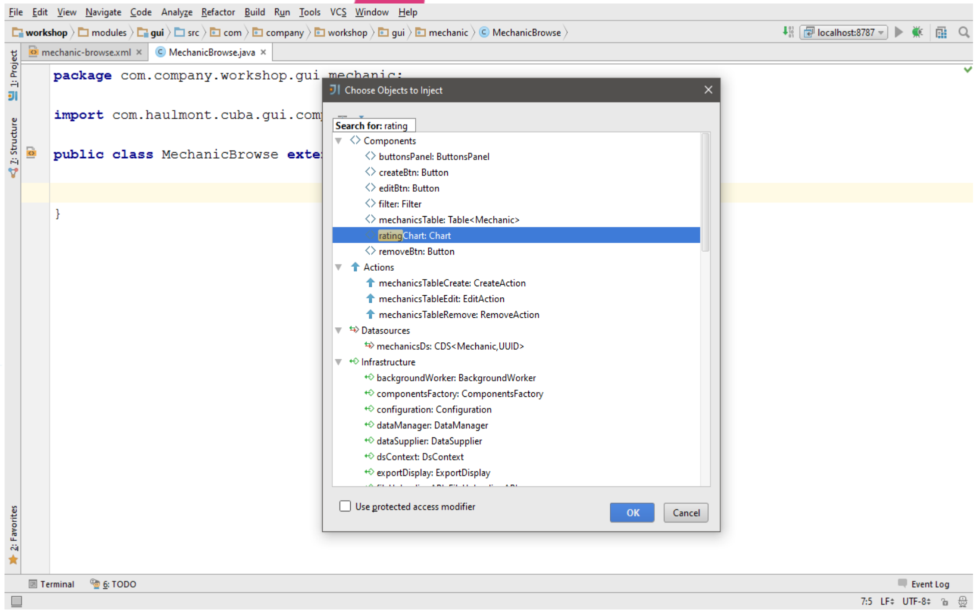
```
@Override
@Transactional
public Map<Mechanic, Long> getMechanicOrdersStats() {
    EntityManager em = persistence.getEntityManager();

    Query query = em.createQuery(
            "select o.mechanic.id, count(o.id) " +
            "from workshop$Order o group by o.mechanic");

    List<Object[]> resultList = query.getResultList();

    Map<Mechanic, Long> stats = new HashMap<>();
    for (Object[] o : resultList) {
        UUID mechanicId = (UUID) o[0];
        Mechanic mechanic = em.find(Mechanic.class,
                mechanicId, View.MINIMAL);

        stats.put(mechanic, (Long)o[1]);
    }
    return stats;
}
```


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
workshop › modules › core › src › com › company › workshop › service › OrderServiceBean
localhost:8787
OrderService.java ×
OrderServiceBean.java ×
1: Project
Z: Structure
2: Favorites
Terminal
6: TODO
Event Log
59:6 LF÷ UTF-8÷

```
    @Override
    public BigDecimal calculateAmount(Order order) {...}

    @Override
    @Transactional
    public Map<Mechanic, Long> getMechanicOrdersStats() {
        EntityManager em = persistence.getEntityManager();

        Query query = em.createQuery(
                "select o.mechanic.id, count(o.id) " +
                "from workshop$Order o group by o.mechanic");

        List<Object[]> resultList = query.getResultList();

        Map<Mechanic, Long> stats = new HashMap<>();
        for (Object[] o : resultList) {
            UUID mechanicId = (UUID) o[0];
            Mechanic mechanic = em.find(Mechanic.class,
                    mechanicId, View.MINIMAL);

            stats.put(mechanic, (Long)o[1]);
        }
        return stats;
    }
}
```

Inject chart component to a screen

1. Go to the ***mechanic-browse*** screen using the Studio **IDE** button
2. Open Java controller (***MechanicBrowse*** class)
3. Use **Alt-Insert** shortcut to inject ***ratingChart*** object to the controller


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
workshop > modules > gui > src > com > company > workshop > gui > mechanic > MechanicBrowse
localhost:8787
mechanic-browse.xml
MechanicBrowse.java
1: Project
2: Structure
2: Favorites
Terminal
6: TODO
Event Log
7:5 LF: UTF-8:

```
package com.company.workshop.gui.mechanic;

import com.haulmont.cuba.gui.com

public class MechanicBrowse exte

}
```

Choose Objects to Inject

Search for: rating

- Components
  - buttonsPanel: ButtonsPanel
  - createBtn: Button
  - editBtn: Button
  - filter: Filter
  - mechanicsTable: Table<Mechanic>
  - ratingChart: Chart
  - removeBtn: Button
- Actions
  - mechanicsTableCreate: CreateAction
  - mechanicsTableEdit: EditAction
  - mechanicsTableRemove: RemoveAction
- Datasources
  - mechanicsDs: CDS<Mechanic,UUID>
- Infrastructure
  - backgroundWorker: BackgroundWorker
  - componentsFactory: ComponentsFactory
  - configuration: Configuration
  - dataManager: DataManager
  - dataSupplier: DataSupplier
  - dsContext: DsContext
  - exportDisplay: ExportDisplay

☐ Use protected access modifier

OK Cancel

Data binding for chart

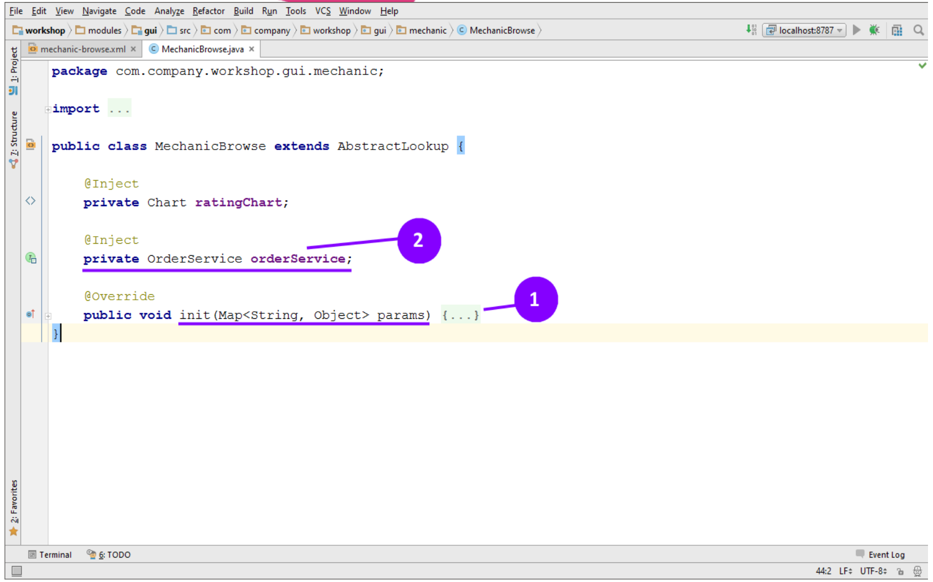
We can connect the chart to data in two ways. The **first way** is to use a data source returning a list of CUBA entities. If we don't have an entity, that describes the content of a chart item we cannot follow this way. The **second way** is to use the ***DataProvider*** interface, which allows us to use arbitrary data in a form that is understood by the chart.

Our data model doesn't have an entity that describes the stats on mechanics, so we'll use the second way.

1. Override the ***init()*** method

   Use **Ctrl-O** to quick override

2. Add ***OrderService*** field with ***@Inject*** annotation

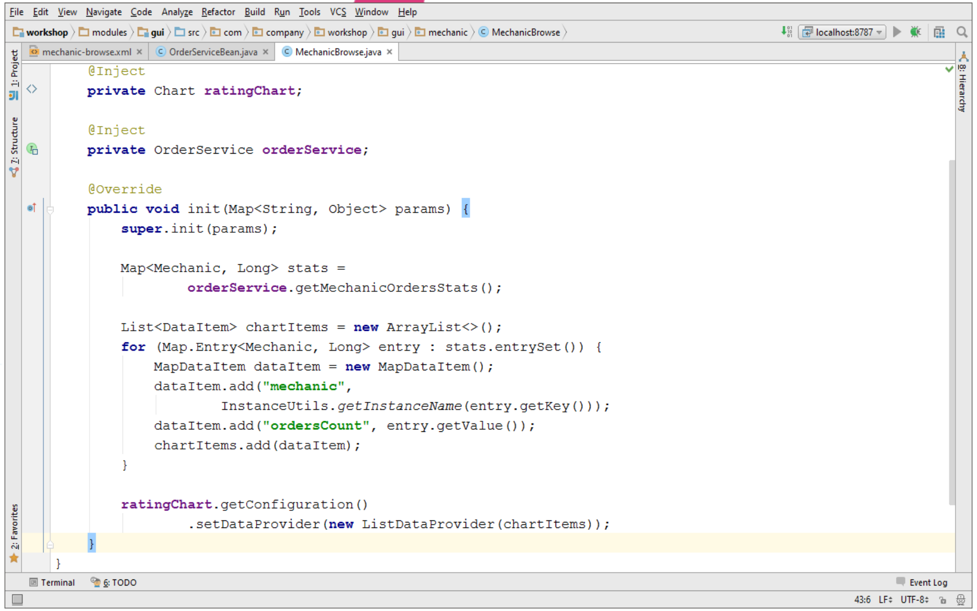


```
package com.company.workshop.gui.mechanic;

import ...

public class MechanicBrowse extends AbstractLookup {

    @Inject
    private Chart ratingChart;

    @Inject
    private OrderService orderService;

    @Override
    public void init(Map<String, Object> params) {...}
}
```

Connect chart with data

Set data to chart using ***ListDataProvider***. The implementation of the ***init(...)*** method is printed below:

```
@Override
public void init(Map<String, Object> params) {
    super.init(params);

    Map<Mechanic, Long> stats =
            orderService.getMechanicOrdersStats();

    List<DataItem> chartItems = new ArrayList<>();
    for (Map.Entry<Mechanic, Long> entry : stats.entrySet()) {
        MapDataItem dataItem = new MapDataItem();
        dataItem.add("mechanic",
                InstanceUtils.getInstanceName(entry.getKey()));
        dataItem.add("ordersCount", entry.getValue());
        chartItems.add(dataItem);
    }

    ratingChart.getConfiguration()
            .setDataProvider(new ListDataProvider(chartItems));
}
```



```
@Inject
private Chart ratingChart;

@Inject
private OrderService orderService;

@Override
public void init(Map<String, Object> params) {
    super.init(params);

    Map<Mechanic, Long> stats =
            orderService.getMechanicOrdersStats();

    List<DataItem> chartItems = new ArrayList<>();
    for (Map.Entry<Mechanic, Long> entry : stats.entrySet()) {
        MapDataItem dataItem = new MapDataItem();
        dataItem.add("mechanic",
                InstanceUtils.getInstanceName(entry.getKey()));
        dataItem.add("ordersCount", entry.getValue());
        chartItems.add(dataItem);
    }

    ratingChart.getConfiguration()
            .setDataProvider(new ListDataProvider(chartItems));
}
}
```

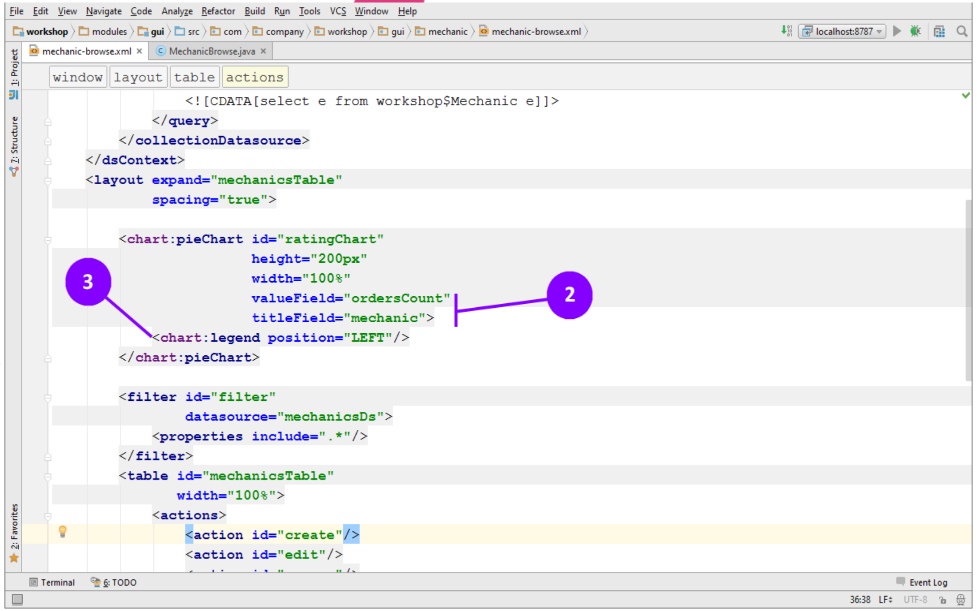
Field mapping for chart

We have connected the data collection, but how will the chart know which fields to use for illustration?

Open ***mechanic-browse.xml*** in the IDE

1. Specify two attributes of the chart: ***valueField*** and ***titleField***. They determine which fields will be used in the chart
2. Add a legend element to set position of the legend for the chart:

```
<chart:pieChart id="ratingChart"
                height="200px"
                width="100%"
                valueField="ordersCount"
                titleField="mechanic">
    <chart:legend position="LEFT"/>
</chart:pieChart>
```


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
workshop modules gui src com company workshop gui mechanic mechanic-browse.xml
localhost:8787
mechanic-browse.xml MechanicBrowse.java
window layout table actions
1: Project
7: Structure
2: Favorites
3
2
Terminal 6: TODO Event Log
36:38 LF UTF-8

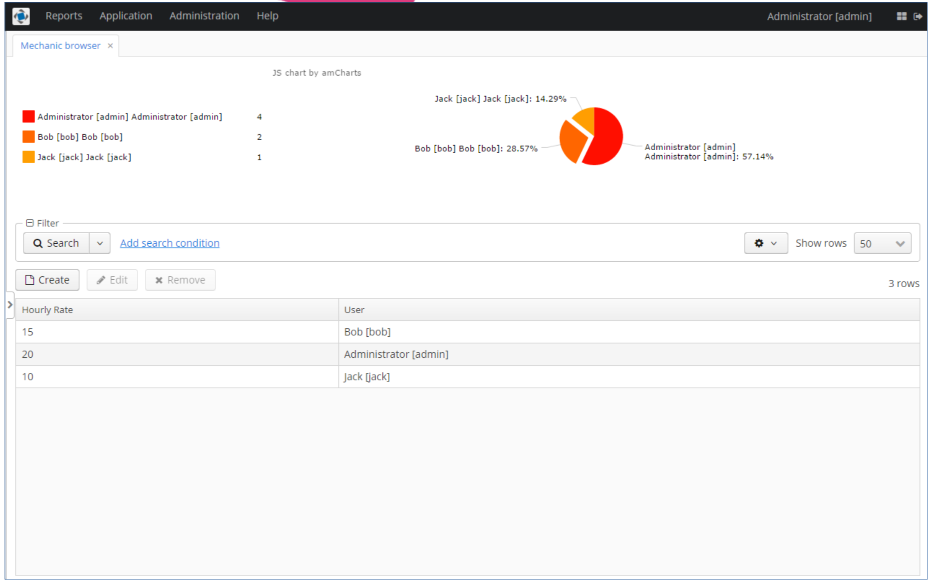
```
                <![CDATA[select e from workshop$Mechanic e]]>
            </query>
        </collectionDatasource>
    </dsContext>
    <layout expand="mechanicsTable"
            spacing="true">

        <chart:pieChart id="ratingChart"
                        height="200px"
                        width="100%"
                        valueField="ordersCount"
                        titleField="mechanic">
            <chart:legend position="LEFT"/>
        </chart:pieChart>

        <filter id="filter"
                datasource="mechanicsDs">
            <properties include=".*"/>
        </filter>
        <table id="mechanicsTable"
               width="100%">
            <actions>
                <action id="create"/>
                <action id="edit"/>
```

CUBA.platform
www.cuba-platform.com

Open screen with chart

1. **Restart** the application using the Studio
2. Open **Application — Mechanics** from the menu

Now we know exactly who should get a bonus.



🟥	Administrator [admin] Administrator [admin]	4
🟧	Bob [bob] Bob [bob]	2
🟨	Jack [jack] Jack [jack]	1

Hourly Rate	User
15	Bob [bob]
20	Administrator [admin]
10	Jack [jack]

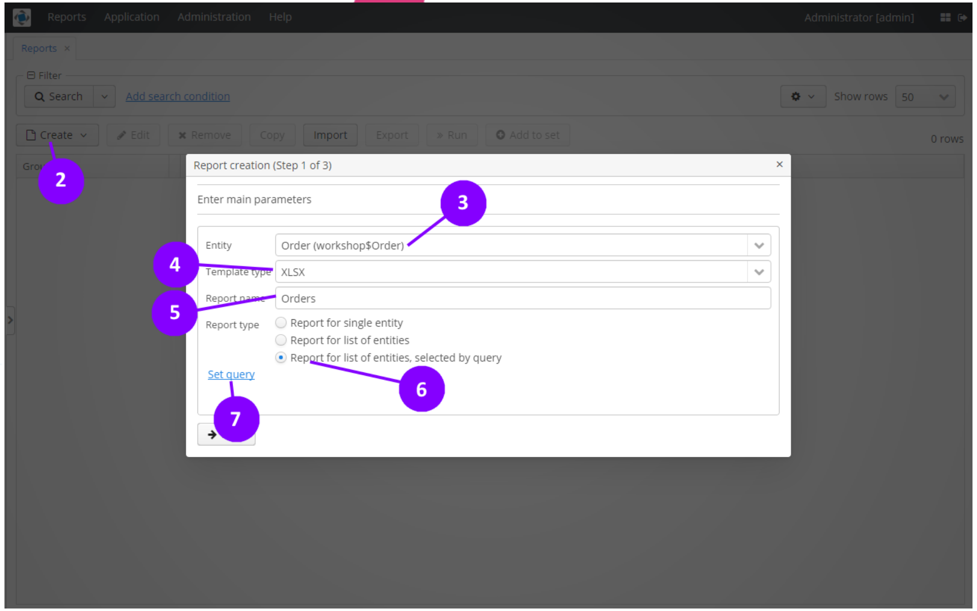
www.cuba-platform.com

Reporting

Reports

A rare business application goes without reports. That’s why our mechanic has asked us to make a report, showing undertaken work for a certain period of time.

1. Open **Reports — Reports** from the menu
2. Click **Create — Using wizard**
3. Select **Entity**: ***Order*** (***workshop$Order***)
4. Set **Template type**: ***XLSX***
5. Set **Report Name**: ***Orders***
6. Select **Report type**: ***Report for list of entities by query***
7. Click **Set query**



Report creation (Step 1 of 3)

Enter main parameters

Entity: Order (workshop$Order)

Template type: XLSX

Report name: Orders

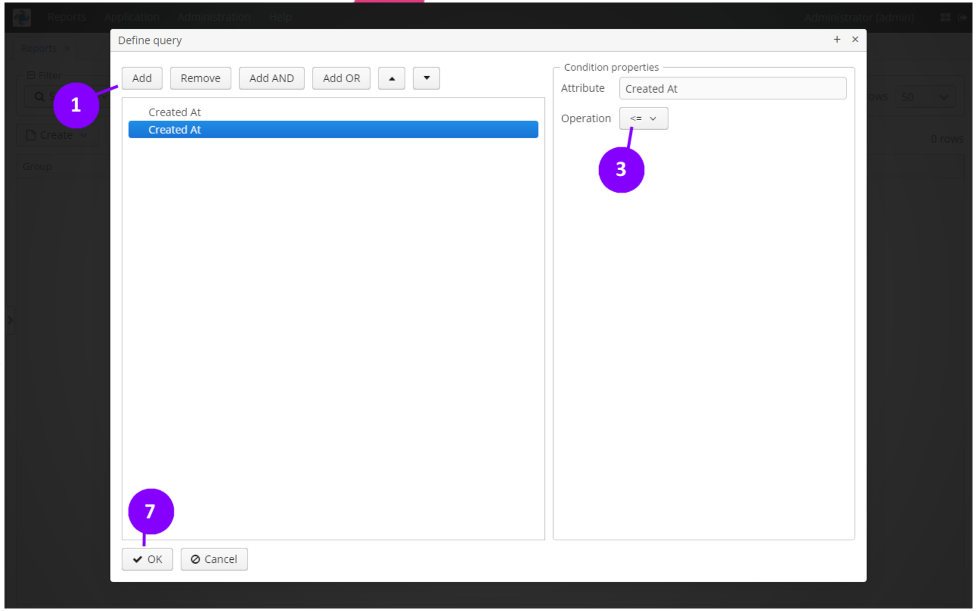
Report type: ○ Report for single entity ○ Report for list of entities [✓ Report for list of entities, selected by query]

Set query

Report query builder

Report Wizard allows us to create a query using the graphical expressions constructor.

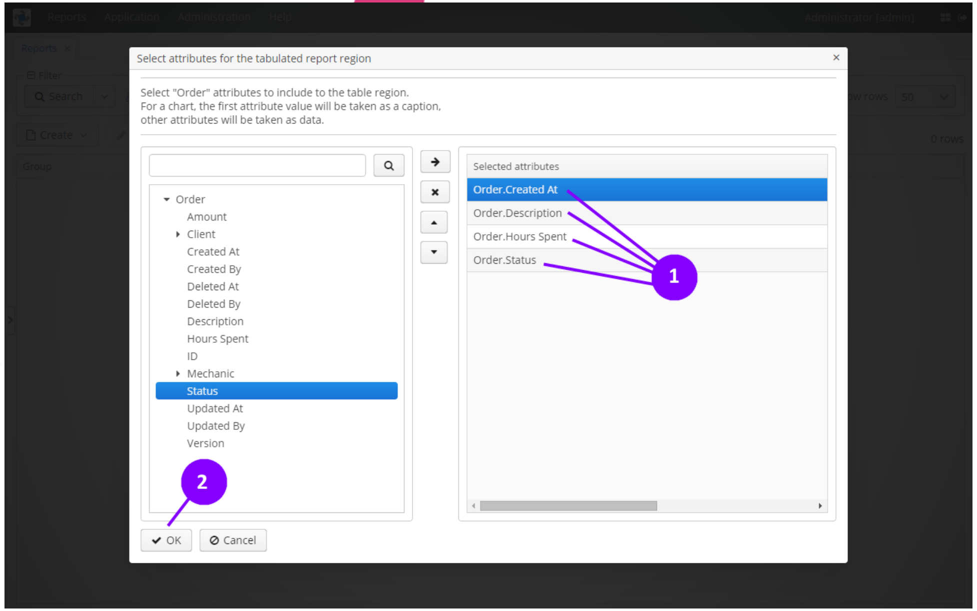
1. Click **Add**
2. Select the **Created at** attribute
3. Change operation for created condition to **[>=]**
4. Click **Add** once again
5. Select the **Created at** attribute
6. Change operation for created condition to **[<=]**
7. Click **OK**
8. Click **Next**


Define query

Condition properties
Attribute: Created At
Operation: <=

Select attributes for report

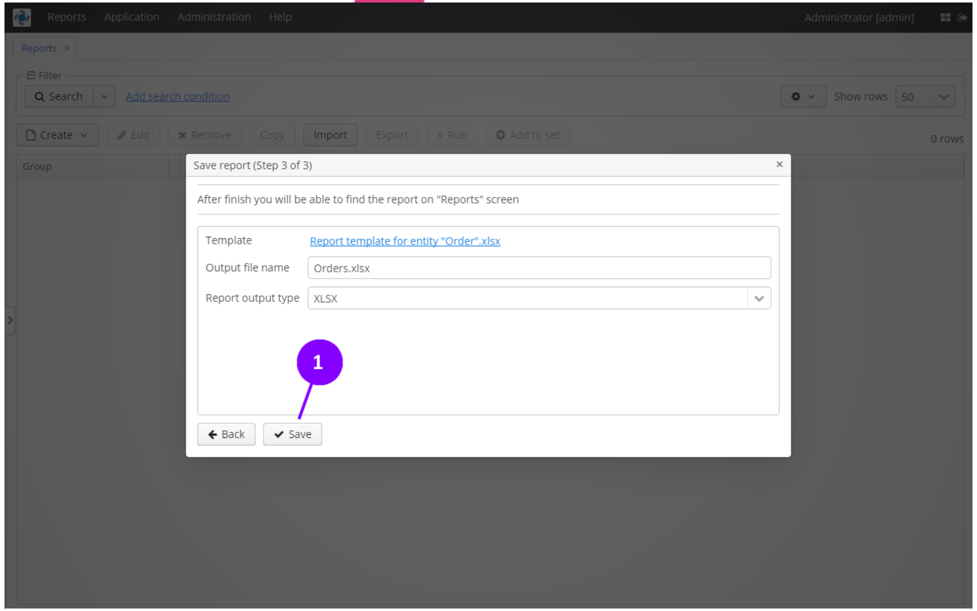
1. Select ***Order*** attributes that the report will contain: ***Created At***, ***Description***, ***Hours Spent***, ***Status***
2. Click **OK**
3. Click **Next**


Select attributes for the tabulated report region

Select "Order" attributes to include to the table region.
For a chart, the first attribute value will be taken as a caption,
other attributes will be taken as data.

Save report

1. Click **Save** to save the report



Save report (Step 3 of 3)

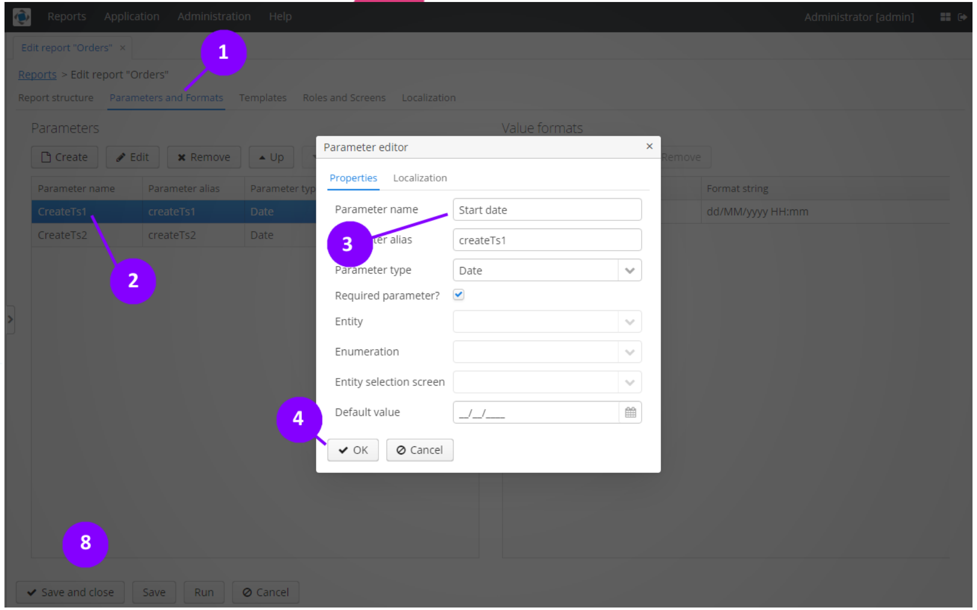
After finish you will be able to find the report on "Reports" screen

Template	Report template for entity "Order".xlsx
Output file name	Orders.xlsx
Report output type	XLSX

Change parameter names

The **Wizard** will open the report editor so that we can make additional changes, if needed.

1. Open **Parameters and Formats** tab
2. Edit the ***CreateTs1*** parameter
3. Set Parameter Name: ***Start date***
4. Click **OK**
5. Edit the ***CreateTs2*** parameter
6. Set Parameter Name: ***End date***
7. Click **OK**
8. Click **Save and close**



Parameter name	Parameter alias	Parameter typ
CreateTs1	createTs1	Date
CreateTs2	createTs2	Date

Format string
dd/MM/yyyy HH:mm

Parameter editor

Properties | Localization

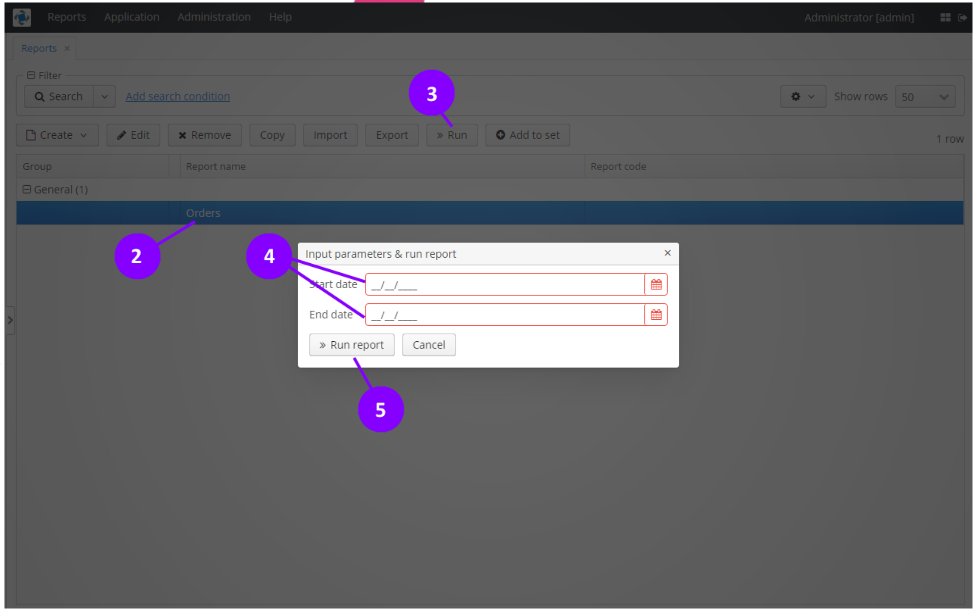
Parameter name	Start date
Parameter alias	createTs1
Parameter type	Date
Required parameter?	✓
Entity	
Enumeration	
Entity selection screen	
Default value	__/__/____

OK | Cancel

Run report

1. Expand **General** report group
2. Select the report
3. Click **Run**
4. Enter **Start date** and **End date**
5. Click **Run report**

The system has generated an **XSLX file**, we can download it and view its content. Due to the fact that the report templates have the same format as the one that is required for the output, we can easily prepare templates from customer's report examples.



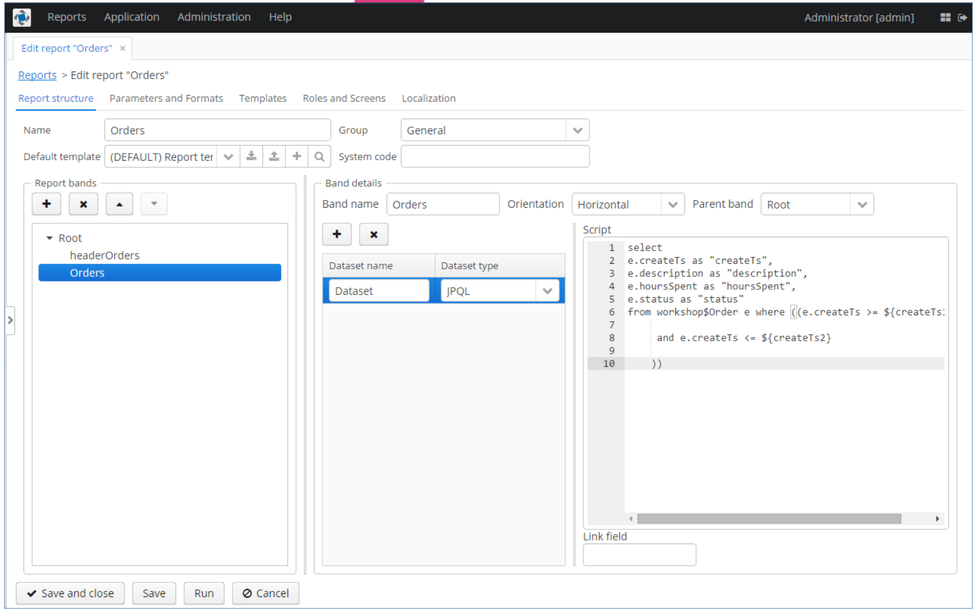
Report editor

You can also create reports manually using the ***Report editor***. Data can be extracted via ***SQL***, ***JPQL*** or even ***Groovy*** scripts.

The template is created in ***XLS(X)***, ***DOC(X)***, ***ODT***, ***HTML*** formats using standard tools.

Report output can be converted to ***PDF***.

Also using the **Report editor** you can specify users who will have access to the report, and system screens where it should appear.



Name	Orders	Group	General
Default template	(DEFAULT) Report ter	System code	

Band name	Orders	Orientation	Horizontal	Parent band	Root

Dataset name	Dataset type
Dataset	JPQL

```
select
e.createTs as "createTs",
e.description as "description",
e.hoursSpent as "hoursSpent",
e.status as "status"
from workshop$Order e where ((e.createTs >= ${createTs

        and e.createTs <= ${createTs2}

      ))
```

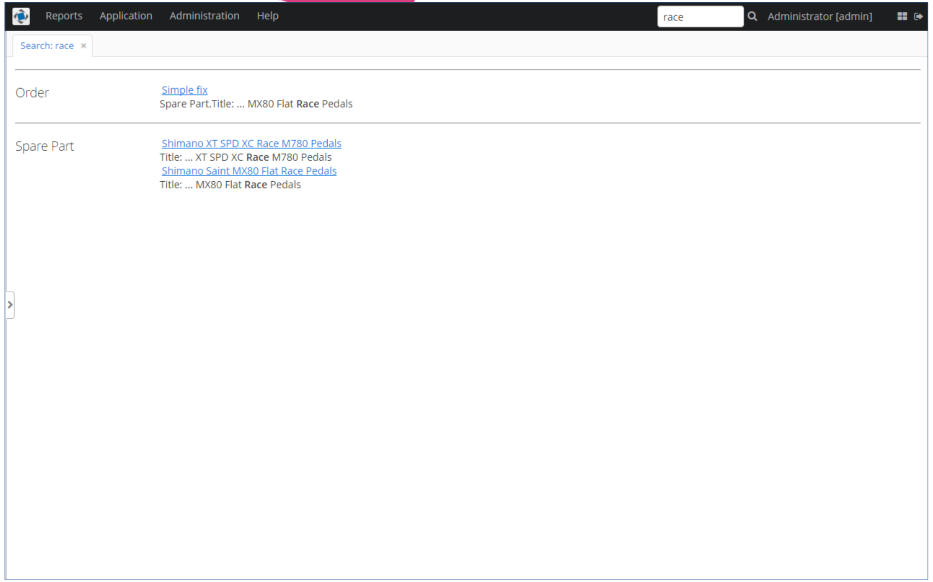
www.cuba-platform.com

Full Text Search

Full Text Search

Our system stores information about spare parts, but there are quite a few of them. It would be useful to search them simply by typing a string like we google in a browser.

The CUBA Platform includes the **Full Text Search** module based on Apache Lucene. It indexes content, including files of different formats, and enables text search using this index.
Search results are filtered according to security constraints.



Order

Simple fix
Spare Part.Title: ... MX80 Flat **Race** Pedals

Spare Part

Shimano XT SPD XC Race M780 Pedals
Title: ... XT SPD XC **Race** M780 Pedals
Shimano Saint MX80 Flat Race Pedals
Title: ... MX80 Flat **Race** Pedals

CUBA.platform
www.cuba-platform.com

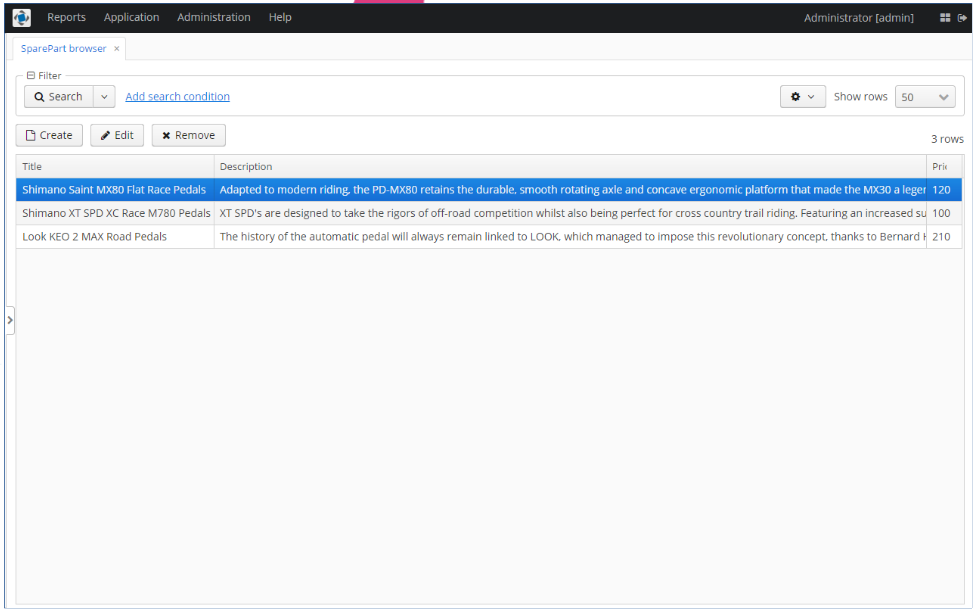
Adding spare parts

1. Open **Application — Spare Parts** from the menu
2. Add spare parts:

   Shimano Saint MX80 Flat Race Pedals

   Shimano XT SPD XC Race M780 Pedals

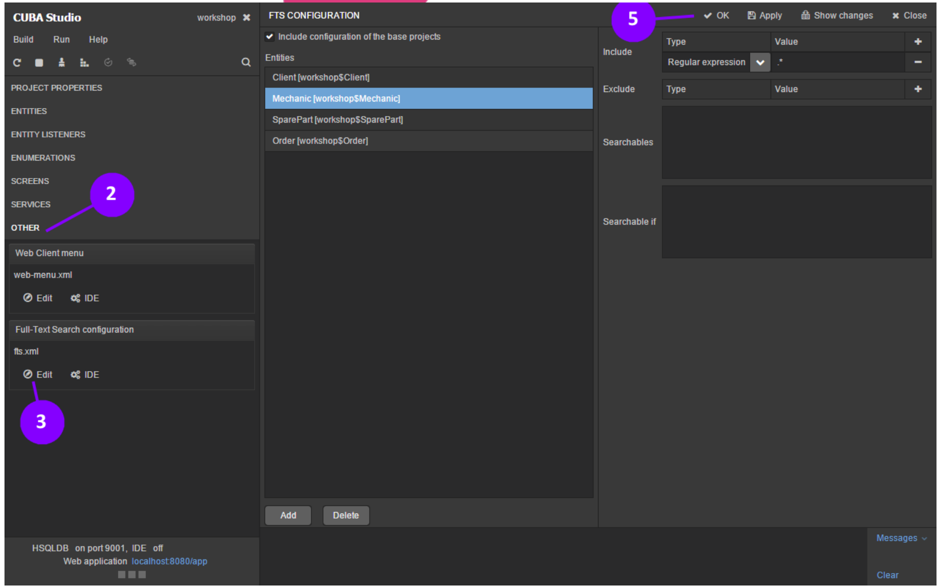
   Look KEO 2 MAX Road Pedals
3. Add these spare parts to random orders



Title	Description	Pri
Shimano Saint MX80 Flat Race Pedals	Adapted to modern riding, the PD-MX80 retains the durable, smooth rotating axle and concave ergonomic platform that made the MX30 a leger	120
Shimano XT SPD XC Race M780 Pedals	XT SPD's are designed to take the rigors of off-road competition whilst also being perfect for cross country trail riding. Featuring an increased su	100
Look KEO 2 MAX Road Pedals	The history of the automatic pedal will always remain linked to LOOK, which managed to impose this revolutionary concept, thanks to Bernard H	210

Configure Full Text Search Index

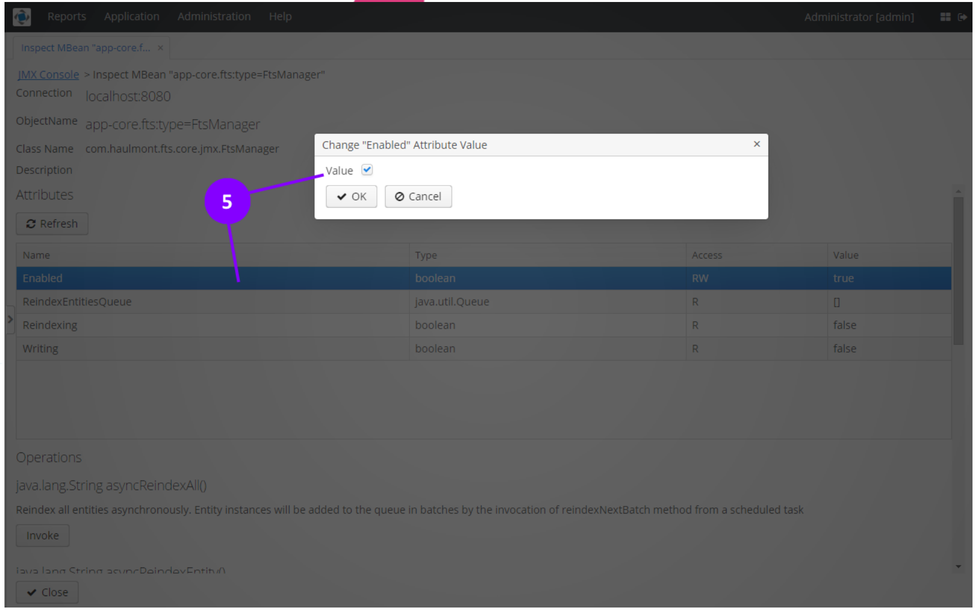
1. Open the Studio
2. Go to **Others** section of the navigation panel
3. Click **Edit** for **Full-Text Search configuration**
4. By default, the Studio has added all our entities to the index configuration. From this screen we can manage entities and fields that will be indexed
5. Click **OK**



Enable Full Text Search for the application

Further configuration will be done via the CUBA interface.

1. Open **Administration — JMX Console** from the menu
2. This is a web version of the console for the JMX interface; it allows us to manage internal system mechanisms
3. Find ***FtsManager*** using the **Search by ObjectName** field
4. Open ***FtsManager***
5. Change the **Enabled** property to true



JMX Console > Inspect MBean "app-core.fts:type=FtsManager"

Connection: localhost:8080

ObjectName: app-core.fts:type=FtsManager

Class Name: com.haulmont.fts.core.jmx.FtsManager

Description

Change "Enabled" Attribute Value

Value ✓

Attributes

Name	Type	Access	Value
Enabled	boolean	RW	true
ReindexEntitiesQueue	java.util.Queue	R	[]
Reindexing	boolean	R	false
Writing	boolean	R	false

Operations

java.lang.String asyncReindexAll()

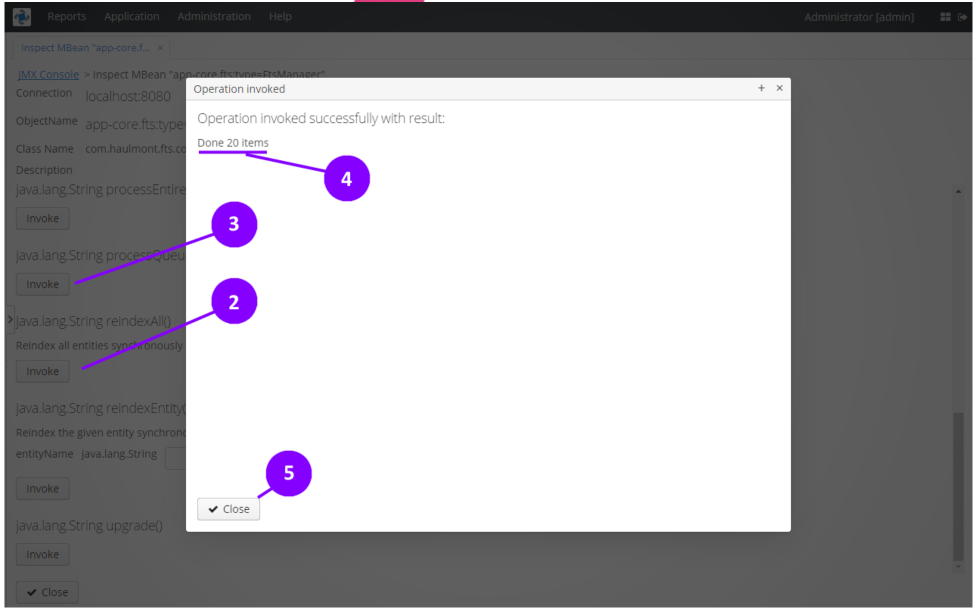
Reindex all entities asynchronously. Entity instances will be added to the queue in batches by the invocation of reindexNextBatch method from a scheduled task

java.lang.String asyncReindexEntity()

CUBA.platform
www.cuba-platform.com

Add records to index

1. Scroll down to see ***reindexAll*** and ***processQueue*** methods of ***FtsManager***
2. Invoke the ***FtsManager reindexAll()*** method
3. Invoke the ***FtsManager processQueue()*** method
4. The system will display the current indexed number of records
5. Click **Close**



JMX Console > Inspect MBean "app-core.fts:type=FtsManager"

Connection localhost:8080

ObjectName app-core.fts:type

Class Name com.haulmont.fts.co

Description

java.lang.String processEntire

Invoke

java.lang.String processQueu

Invoke

java.lang.String reindexAll()

Reindex all entities synchronously

Invoke

java.lang.String reindexEntity(

Reindex the given entity synchron

entityName java.lang.String

Invoke

java.lang.String upgrade()

Invoke

✓ Close

Operation invoked

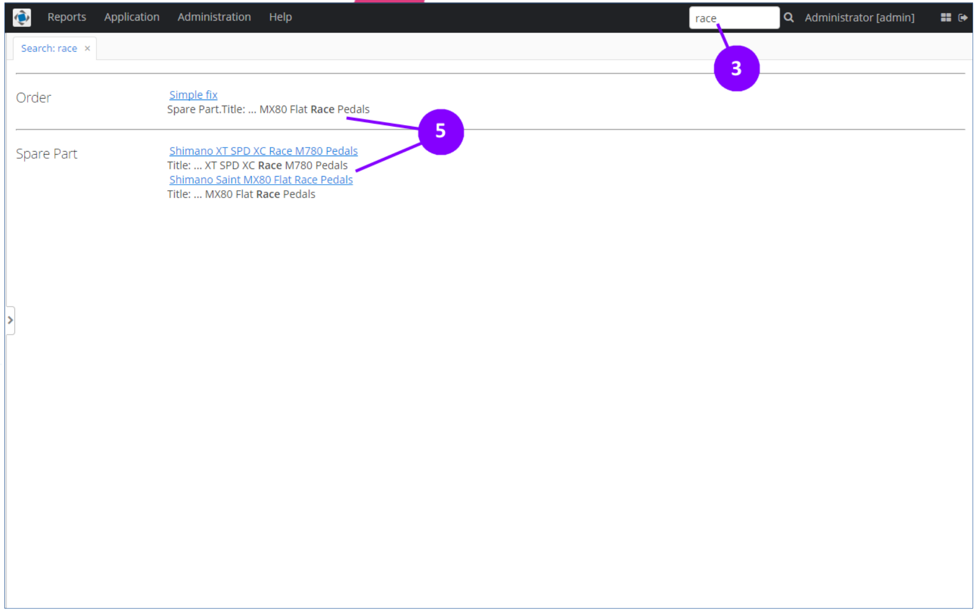
Operation invoked successfully with result:

Done 20 items

✓ Close

FTS in action

1. **Log out** from the system
2. **Log in** again
3. In the application top panel, the **search field** will appear, allowing you to search through all added to FTS objects
4. Let's find something, for example: ***race***
5. You will see the screen with search results, which contains not only spare parts, but also orders that have spare parts with this word in its name



Order
Simple fix
Spare Part.Title: ... MX80 Flat **Race** Pedals

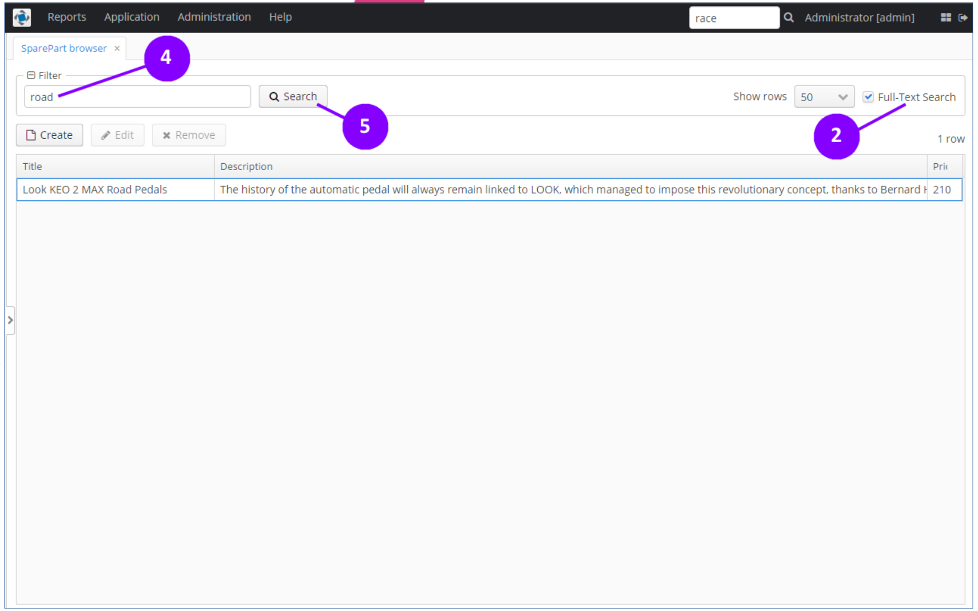
Spare Part
Shimano XT SPD XC Race M780 Pedals
Title: ... XT SPD XC **Race** M780 Pedals
Shimano Saint MX80 Flat Race Pedals
Title: ... MX80 Flat **Race** Pedals

FTS integration with filters

But what if we want to search only for spare parts?

1. Open **Application — Spare Parts** from the menu
2. Select **Full Text Search** checkbox in the filter panel
3. The text field will appear
4. Let's enter something, for example: ***road***
5. Click **Search**
6. The table will display records that contain ***road*** in their ***description***

So, now mechanics will be able to find spare parts by description quickly



Title	Description	Pri
Look KEO 2 MAX Road Pedals	The history of the automatic pedal will always remain linked to LOOK, which managed to impose this revolutionary concept, thanks to Bernard H	210

Audit

Audit

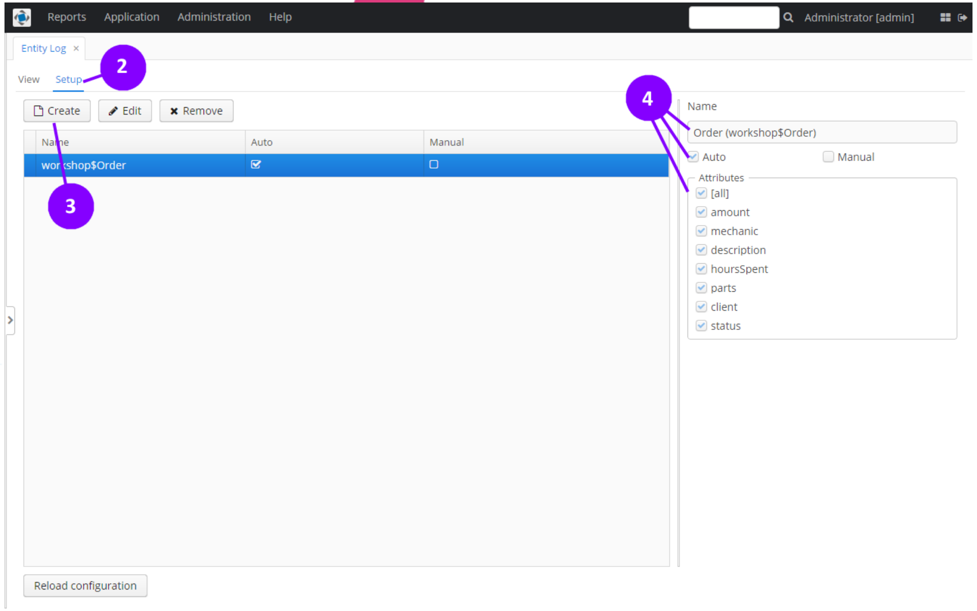
It happens when one day someone has accidentally erased the order description. It is not appropriate to call the client on the phone, apologize and ask them to repeat the what needs to be done. Let's see how this can be avoided. CUBA has a built-in mechanism to track entity changes, which you can configure to track operations with critical system data.

Let's apply it for logging order changes.

1. Open **Administration — Entity log** from the menu
2. Go to the **Setup** tab
3. Click **Create**
4. Set **Name**: ***Order*** (***workshop$Order***)

   Auto: **true**

   Attributes: ***all***
5. Click **Save**



Name	Auto	Manual
workshop$Order	✓	☐

Name: Order (workshop$Order)

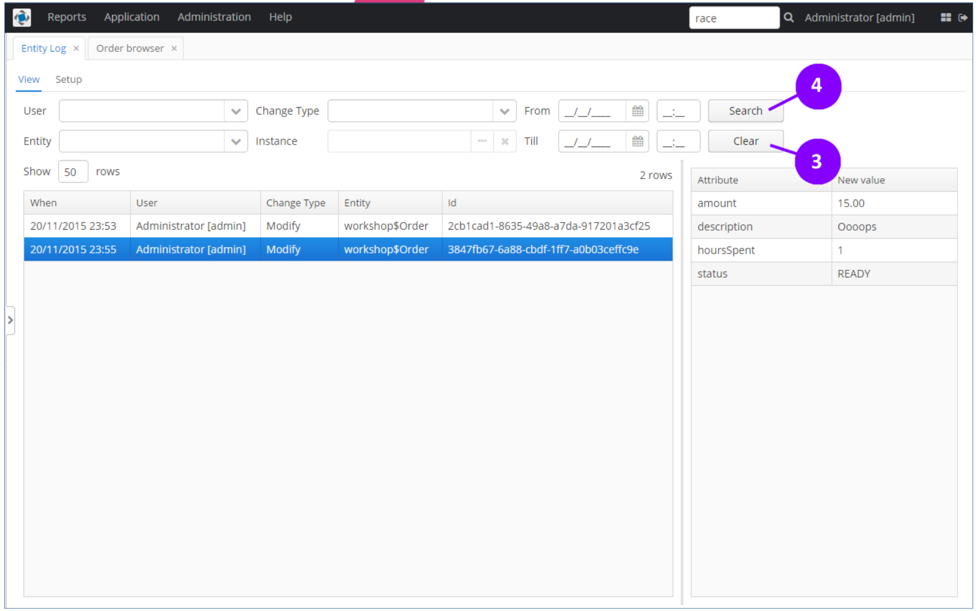
[✓] Auto ☐ Manual

Attributes
[✓] [all]
[✓] amount
[✓] mechanic
[✓] description
[✓] hoursSpent
[✓] parts
[✓] client
[✓] status

Audit in action

1. Let's change an order description (or even clean it up)
2. Open **Administration — Entity Log**
3. Click **Clear** to reset security records filter
4. Click **Search**

The table shows changes and the user that made them, the changed fields and their new values. By sorting the changes by date and filtering them for a particular record, we'll be able to restore the entire chronology of events.



When	User	Change Type	Entity	Id
20/11/2015 23:53	Administrator [admin]	Modify	workshop$Order	2cb1cad1-8635-49a8-a7da-917201a3cf25
20/11/2015 23:55	Administrator [admin]	Modify	workshop$Order	3847fb67-6a88-cbdf-1ff7-a0b03ceffc9e

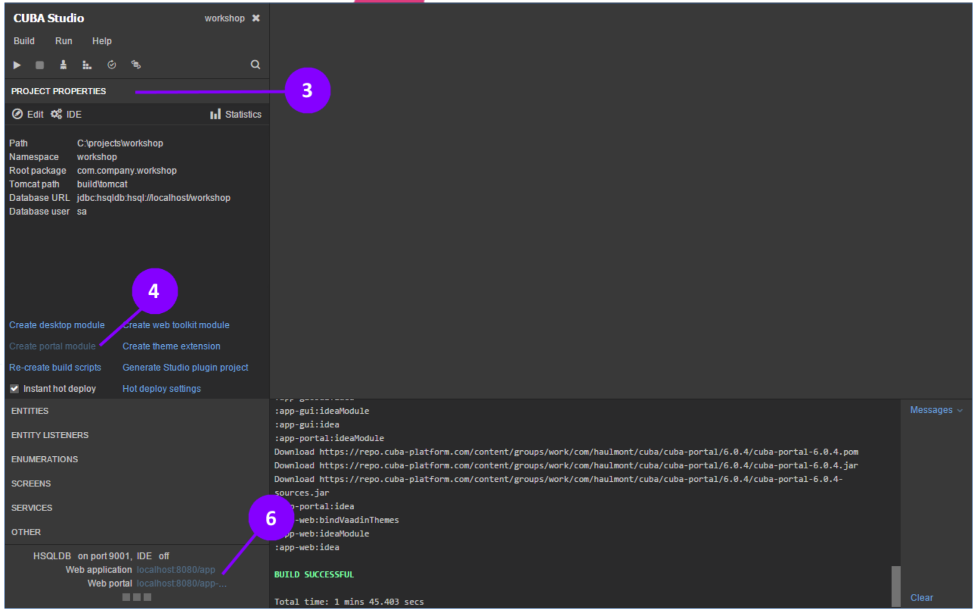
Attribute	New value
amount	15.00
description	Oooops
hoursSpent	1
status	READY

REST-API

Portal module

Let's try to add one more type of interface to our project - web portal. Similar to the web client, a portal can be deployed separately from the middle tier. Similar to the web client, it will have access to the middle layer services, even in distributed configuration. The portal is intended for a customer faced clients such as mobile devices or fancy web pages.

1. **Open** the Studio
2. **Stop** the application
3. Go to the **Project properties** section
4. Click the **Create portal module** link
5. Confirm action by clicking **OK**
6. At the bottom of the Studio window, we'll see a new link to the Web portal page



Path	C:\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 off
Web application localhost:8080/app
Web portal localhost:8080/app-...

```
:app-gui:ideaModule
:app-gui:idea
:app-portal:ideaModule
Download https://repo.cuba-platform.com/content/groups/work/com/haulmont/cuba/cuba-portal/6.0.4/cuba-portal-6.0.4.pom
Download https://repo.cuba-platform.com/content/groups/work/com/haulmont/cuba/cuba-portal/6.0.4/cuba-portal-6.0.4.jar
Download https://repo.cuba-platform.com/content/groups/work/com/haulmont/cuba/cuba-portal/6.0.4/cuba-portal-6.0.4-
sources.jar
  p-portal:idea
   -web:bindVaadinThemes
  pp-web:ideaModule
:app-web:idea

BUILD SUCCESSFUL

Total time: 1 mins 45.403 secs
```

Generic REST API

The portal is a classic Spring MVC application that has access to the entities and services of the main system. A new module, **portal**, will be added to our project. It will have the source code of Spring MVC controllers and configuration files.

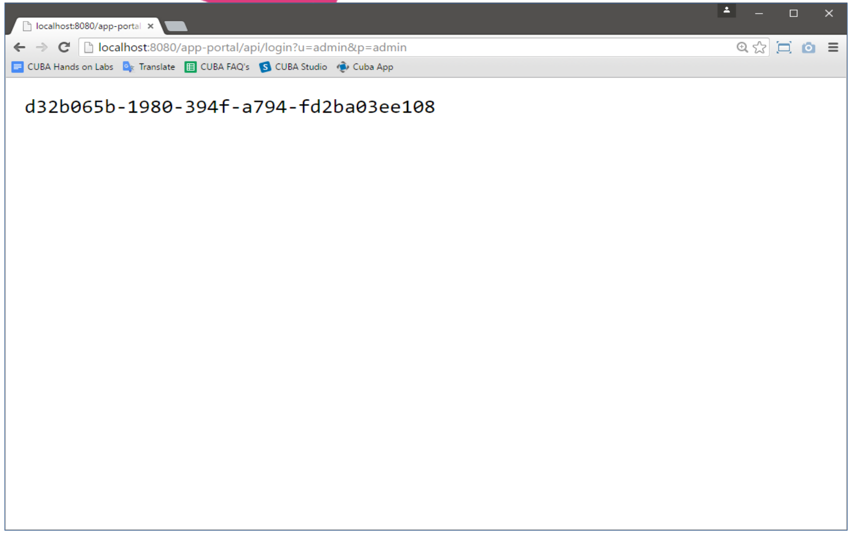
In addition to classic Spring MVC application based on the portal module, you can build AJAX applications that use the REST interface to access the data. The **universal REST-API** of the platform allows to load and save all entities defined in the application data model by sending simple HTTP requests.

This provides an easy way to integrate with a wide range of third-party applications – from the JavaScript code, executed in the browser, to mobile applications or arbitrary systems running Java, .NET, PHP or any other platform.

REST API — obtaining session id

1. **Start** application
2. 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. You can invoke the login method right from the browser address bar.

   Try this GET request: http://localhost:8080/app-portal/api/login?u=admin&p=admin


localhost:8080/app-portal
localhost:8080/app-portal/api/login?u=admin&p=admin
CUBA Hands on Labs
Translate
CUBA FAQ's
CUBA Studio
Cuba App

```
d32b065b-1980-394f-a794-fd2ba03ee108
```

REST API — JPQL query

Let’s load the list of new orders in JSON using the following query:

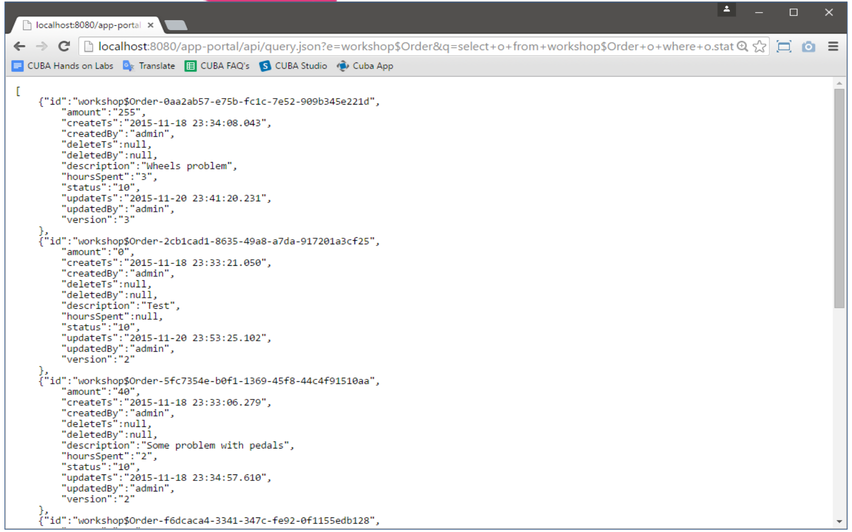
select o from workshop$Order o where o.status = 10

REST-API request:

http://localhost:8080/app-portal/api/query.json?e=workshop$Order&q=select+o+from+workshop$Order+o+where+o.status=10&s=e9c5e533-8c04-4ef9-08c1-8875b2a91ab8

Note: change session id (s parameter) to your actual value

If we change json to xml in the request, then we’ll get the same data in XML. Apart from GET requests, you can use POST for all operations.


localhost:8080/app-portal
localhost:8080/app-portal/api/query.json?e=workshop$Order&q=select+o+from+workshop$Order+o+where+o.stat
CUBA Hands on Labs
Translate
CUBA FAQ's
CUBA Studio
Cuba App

```
[
    {"id":"workshop$Order-0aa2ab57-e75b-fc1c-7e52-909b345e221d",
        "amount":"255",
        "createTs":"2015-11-18 23:34:08.043",
        "createdBy":"admin",
        "deleteTs":null,
        "deletedBy":null,
        "description":"Wheels problem",
        "hoursSpent":"3",
        "status":"10",
        "updateTs":"2015-11-20 23:41:20.231",
        "updatedBy":"admin",
        "version":"3"
    },
    {"id":"workshop$Order-2cb1cad1-8635-49a8-a7da-917201a3cf25",
        "amount":"0",
        "createTs":"2015-11-18 23:33:21.050",
        "createdBy":"admin",
        "deleteTs":null,
        "deletedBy":null,
        "description":"Test",
        "hoursSpent":null,
        "status":"10",
        "updateTs":"2015-11-20 23:53:25.102",
        "updatedBy":"admin",
        "version":"2"
    },
    {"id":"workshop$Order-5fc7354e-b0f1-1369-45f8-44c4f91510aa",
        "amount":"40",
        "createTs":"2015-11-18 23:33:06.279",
        "createdBy":"admin",
        "deleteTs":null,
        "deletedBy":null,
        "description":"Some problem with pedals",
        "hoursSpent":"2",
        "status":"10",
        "updateTs":"2015-11-18 23:34:57.610",
        "updatedBy":"admin",
        "version":"2"
    },
    {"id":"workshop$Order-f6dcaca4-3341-347c-fe92-0f1155edb128",
```

Summary

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 big strong solution.

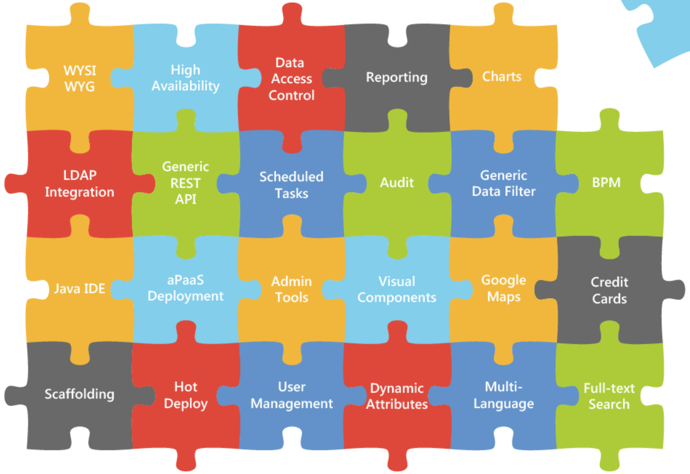
CUBA.platform
www.cuba-platform.com

We have many more features!

In this session covers just a few features of CUBA, but the platform has many more…

If you want to learn more about additional modules and components just take a look at CUBA documentation:

https://www.cuba-platform.com/manual



CUBA.platform
www.cuba-platform.com

Questions?

Visit our forum

https://www.cuba-platform.com/support