



# **Система управления документами и задачами ТЕЗИС™**

*Руководство по разработке расширений*



## Содержание

Предисловие .....	5
1.Назначение .....	5
2.Целевая аудитория .....	5
3.Структура руководства .....	5
4.Соглашения .....	6
4.1.Типографские соглашения .....	6
4.2.Врезки .....	6
4.3.Замечания и предупреждения .....	7
5.Исходные тексты программ .....	7
1.Установка и настройка инструментария .....	8
1.1.Установка Postgres 8.3.9 .....	8
1.2.Установка Java SE 6 .....	17
1.3.Установка Gradle 1.0-milestone-3 .....	25
1.4.Установка IntelliJ IDEA 11 Community Edition .....	25
2.Создание и запуск проекта расширения для системы ТЕЗИС .....	33
2.1.Структура и назначение каталогов и файлов проекта .....	36
3.Быстрый старт .....	40
3.1.Создание таблицы в базе данных .....	40
3.2.Создание класса сущности .....	42
3.3.Создание экрана со списком сущностей .....	46
3.4.Создание экрана редактирования сущности .....	51
3.5.Расширение существующей функциональности .....	54
4.Разработка собственного проекта расширения .....	61
4.1.Создание нового справочника .....	63
4.1.1.Создание таблицы "Статьи бюджета" в БД .....	63
4.1.2.Создание класса сущности BudgetItem .....	65
4.1.3.Создание экранов .....	69
4.2.Создание новой карточки .....	78
4.2.1.Создание таблицы "Счет на оплату" .....	78
4.2.2.Создание класса сущности InvoiceForPayment .....	80
4.2.3.Создание экранов .....	82
4.3.Создание нумератора .....	99
4.4.Создание ролей .....	100
4.5.Редактирование групп доступа .....	108
4.6.Создание процесса оплаты счета .....	111
4.7.Создание папок приложения .....	121
4.7.1.Создание папок приложения, привязанных к процессу .....	129

А.Часто задаваемые вопросы .....	147
Основные определения и понятия .....	153

## Предисловие

### 1. Назначение

Документ является руководством по разработке расширений для системы **ТЕЗИС**. *Расширение* – это подключаемое дополнение, добавляющее новые или изменяющее существующие функциональные возможности системы. Расширение позволяет адаптировать систему к требованиям заказчика, при этом сохраняя возможность последующего обновления версий системы. Информация, представленная в руководстве, актуальна для системы **ТЕЗИС** версии 3.2.

### 2. Целевая аудитория

Руководство предназначено для разработчиков, которые заинтересованы в разработке собственных расширений для системы управления документами и задачами **ТЕЗИС**.

Для успешной работы требуется хорошее знание следующих технологий:

- Java Standard Edition;
- XML;
- SQL.

Перед началом работы рекомендуется ознакомиться с **Руководством пользователя** и с **Руководством администратора** системы **ТЕЗИС**.

### 3. Структура руководства

Данное руководство состоит из следующих разделов:

#### **Глава 1. Установка и настройка инструментария**

В данной главе приводится список программного обеспечения, необходимого для создания расширений. Приведены инструкции по установке и настройке инструментов.

#### **Глава 2. Создание и запуск проекта расширения для системы ТЕЗИС**

Данная глава содержит сведения по созданию шаблонного проекта, в котором в дальнейшем можно будет вести разработку новой функциональности.

## Глава 3. Быстрый старт

В данной главе представлены основные сведения по созданию простой сущности системы **ТЕЗИС**. Информация изложена в краткой форме для того, чтобы Вы могли в кратчайшие сроки овладеть основами создания расширений к системе **ТЕЗИС**.

## Глава 4. Разработка собственного проекта расширения

В данной главе приводятся инструкции по разработке собственного проекта расширения на основе практического примера.

### 4. Соглашения

Для привлечения внимания к определенной информации в Руководстве используется выделение текста различными шрифтами, стилями и размером.

#### 4.1. Типографские соглашения

Далее приводятся примеры выделенных слов и пояснения, в каких случаях используется то или иное оформление.

**Переменные среды, команды, имена объектов Java, тексты ошибок, имена файлов, пакетов.** Обозначены в Руководстве моноширинным шрифтом.

**Элементы интерфейса.** Пункты меню, названия окон, папок, вкладок выделяются в тексте **полужирным текстом**. Кнопки выглядят следующим образом: **Кнопка**.

**Имена приложений, объекты базы данных, названия внешних документов.** Обозначаются в тексте Руководства **полужирным текстом**.

**Параметры и свойства.** *Параметры* и *свойства* также выделяются в тексте Руководства.

#### 4.2. Врезки

Листинг исходного кода, а также содержимое файлов визуально отделяются от окружающего текста следующим образом:

```
[appdefaults]
autologin = true
forward = true
forwardable = true
encrypt = true
```

## 4.3. Замечания и предупреждения

В тексте Руководства будут встречаться сообщения, используемые для привлечения внимания к информации.

### Совет

Это сообщение, содержащее информацию о способах упрощения работы.

### Подсказка

Это дополнительная информация к задаче или проблеме, позволяющая найти правильное решение.

### Внимание

Данные сообщения акцентируют внимание на вещах, которые легко упустить из виду.

## 5. Исходные тексты программ

Все исходные тексты проекта, рассмотренные в руководстве, доступны для ознакомления и находятся в папке `ext-dev\sample\`

## Глава 1. Установка и настройка инструментария

В этой главе описывается процесс установки и настройки рабочей среды разработчика. Для начала работы необходимо установить на локальный компьютер следующие программы:

- **Java SE 6 SDK.**
- **PostgreSQL 8.3.9 .** PostgreSQL – реляционная система управления базами данных. PostgreSQL поставляется под лицензией BSD, что обеспечивает максимальную открытость и доступность.
- **Gradle 1.0-milestone-3 .** Gradle – инструмент автоматизации сборки с открытым исходным кодом.
- **IntelliJ IDEA 11 Community Edition .** IntelliJ IDEA – интегрированная среда разработки для Java компании **JetBrains** . IntelliJ IDEA Community Edition является свободно распространяемой версией с открытым исходным кодом.

### Внимание

Инструкции по установке приведены для операционной системы **Windows 7**

### 1.1. Установка Postgres 8.3.9

Сайт компании-разработчика: <http://www.postgresql.org/>

Дистрибутив: <ftp://ftp-archives.postgresql.org/pub/binary/v8.3.9/>

1. Скачайте архив `postgresql-8.3.9-1.zip` и распакуйте его.
2. Запустите в этом каталоге файл `postgresql-8.3.msi` .
3. В первом окне установите переключатель в положение **Russian** (русский язык установки) и нажмите на кнопку `Start`.

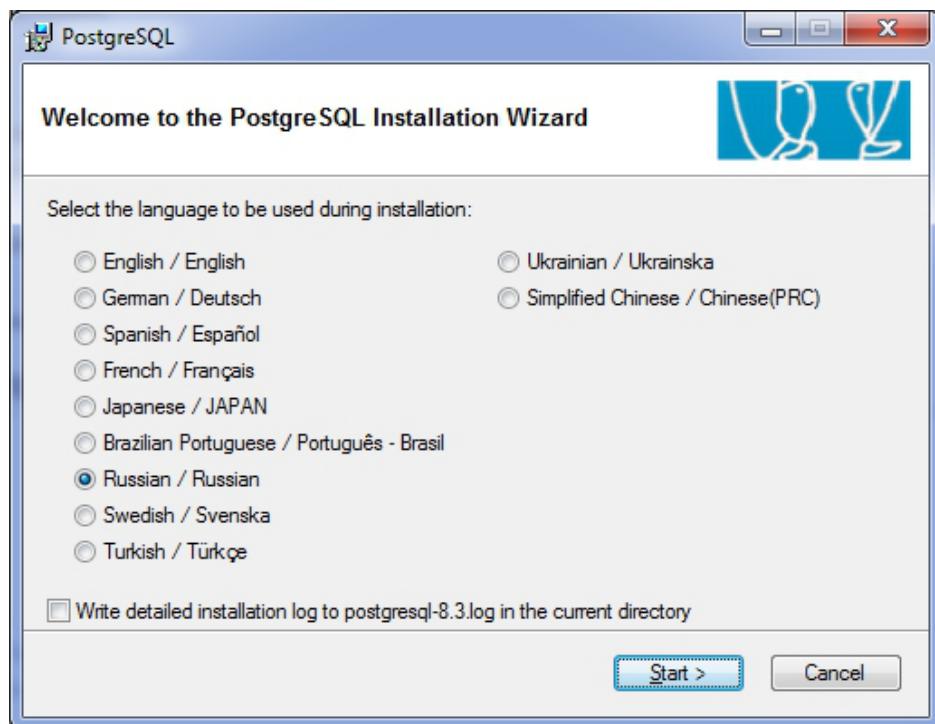


Рисунок 1. Установка Postgres. Шаг 1

4. В следующем окне нажмите на кнопку **Далее**.

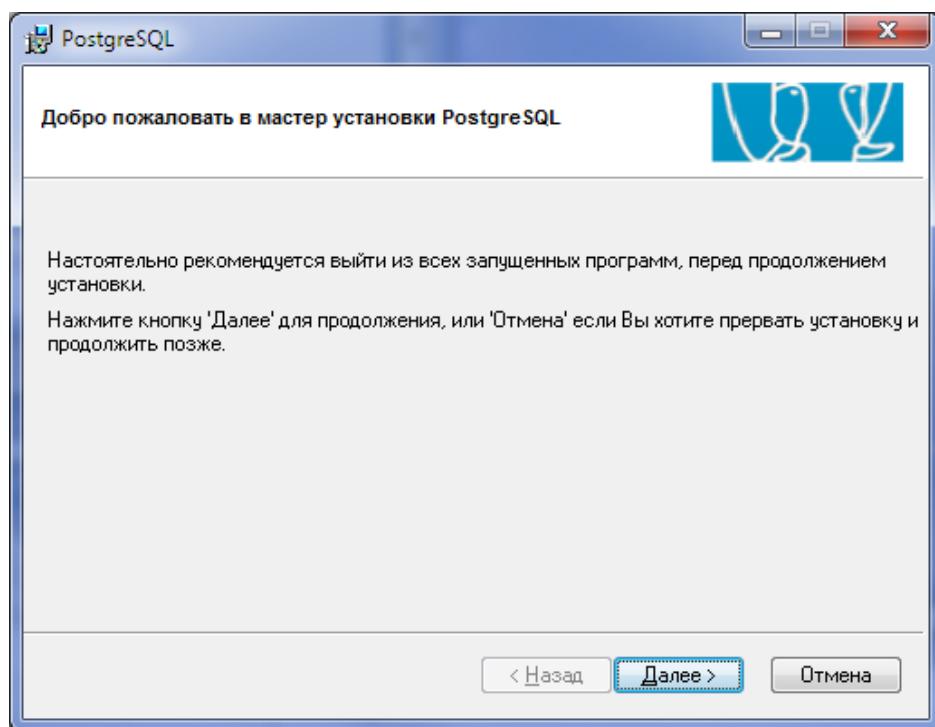


Рисунок 2. Установка Postgres. Шаг 2

5. В следующем окне предоставляется возможность ознакомиться с инструкцией по установке. Нажмите на кнопку **Далее**.

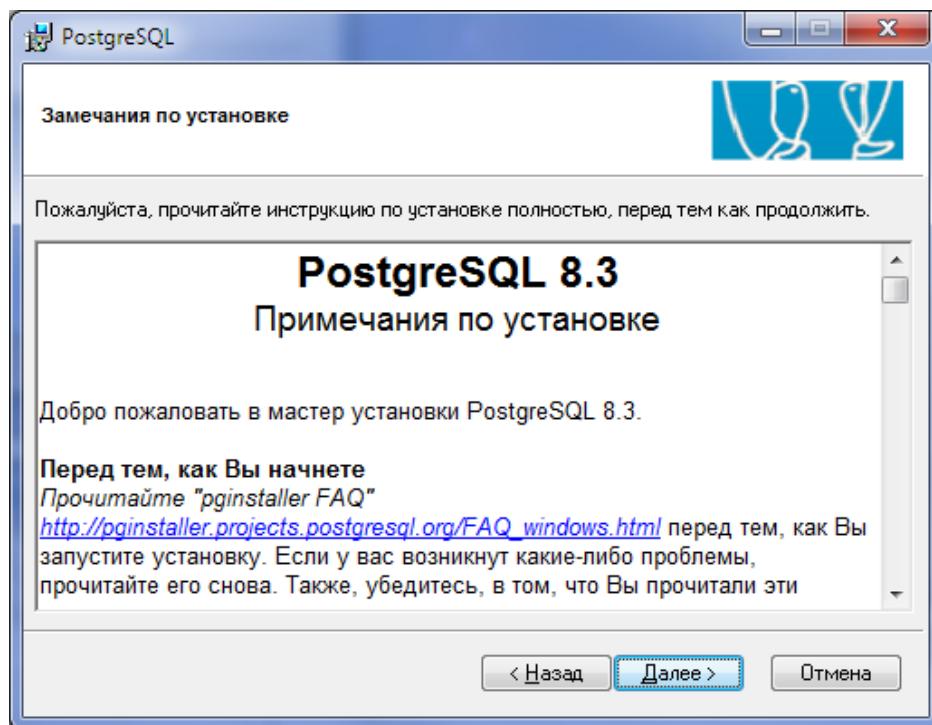


Рисунок 3. Установка Postgres. Шаг 3

6. В следующем окне Вы можете выбрать дополнительные компоненты и сменить каталог установки программы, нажав на кнопку **Обзор** и выбрав новый путь. Нажмите на кнопку **Далее**.

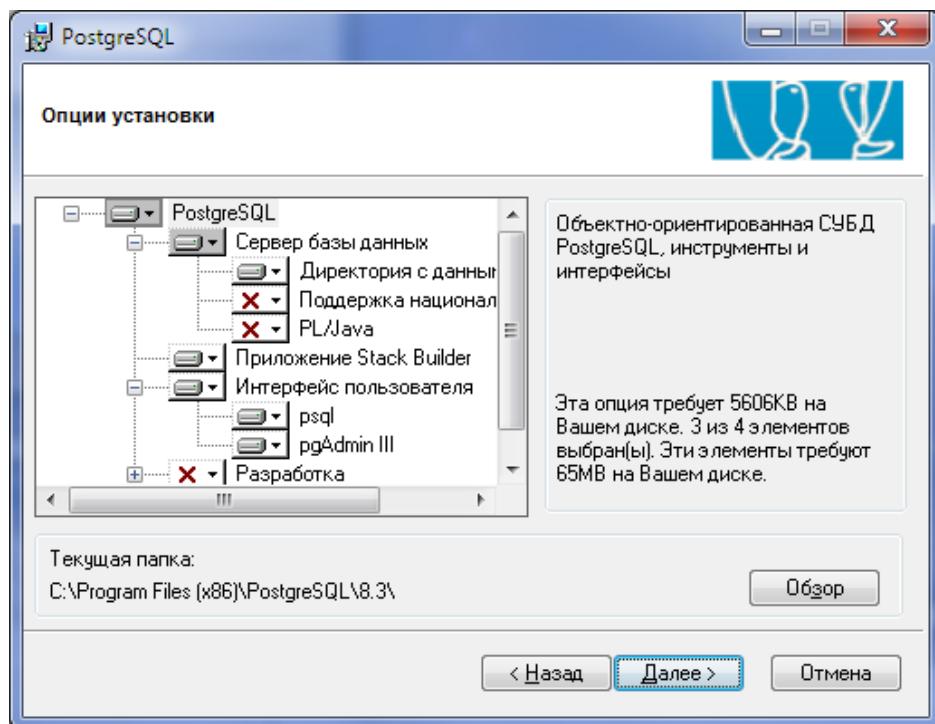


Рисунок 4. Установка Postgres. Шаг 4

7. В следующем окне нужно обязательно убедиться в том, что установлена галочка **Установить как сервис**. В поле ввода **Домен** отображается имя Вашего компьютера. В поля **Пароль** и **Подтверждение** введите пароль, который будет использоваться при входе в учетную запись. Нажмите на кнопку **Далее**.

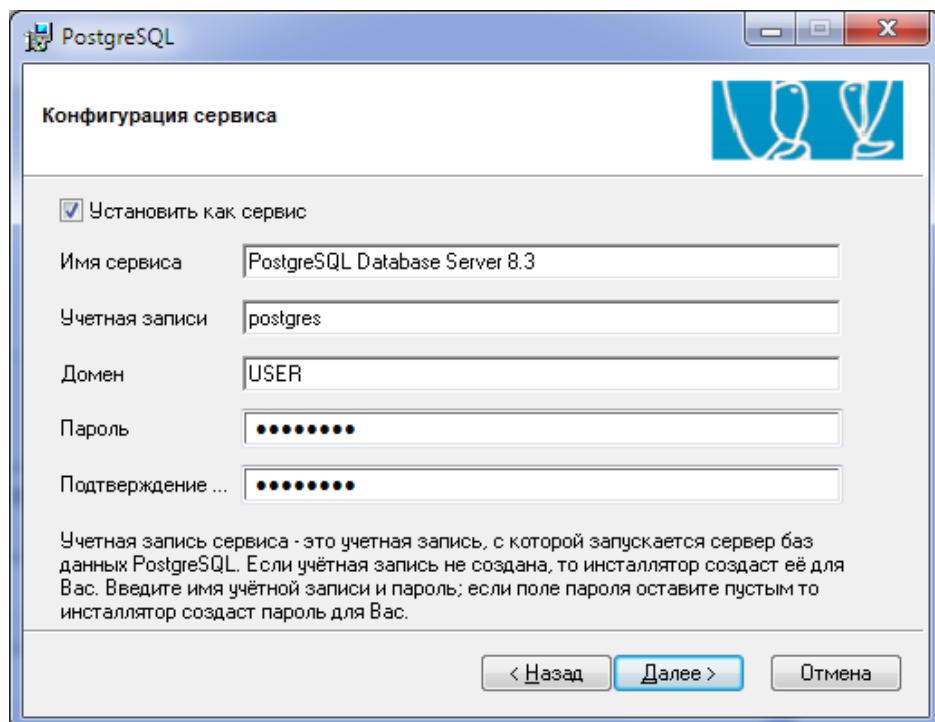


Рисунок 5. Установка Postgres. Шаг 5

8. Если Вы впервые создаете учетную запись, возникнет экран **Ошибка учетной записи**. Если у Вас появился такой экран, нажмите на кнопку **Да**.

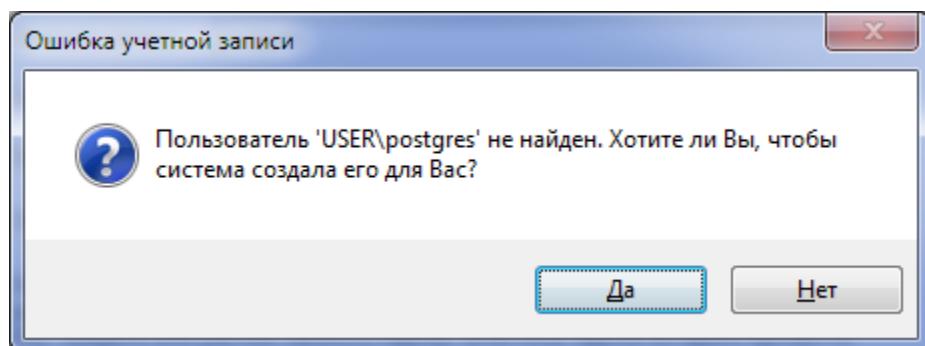


Рисунок 6. Установка Postgres. Окно ошибки учетной записи

9. Если на следующем шаге возникнет окно с предложением заменить Ваш пароль, нажмите на кнопку **Нет**.

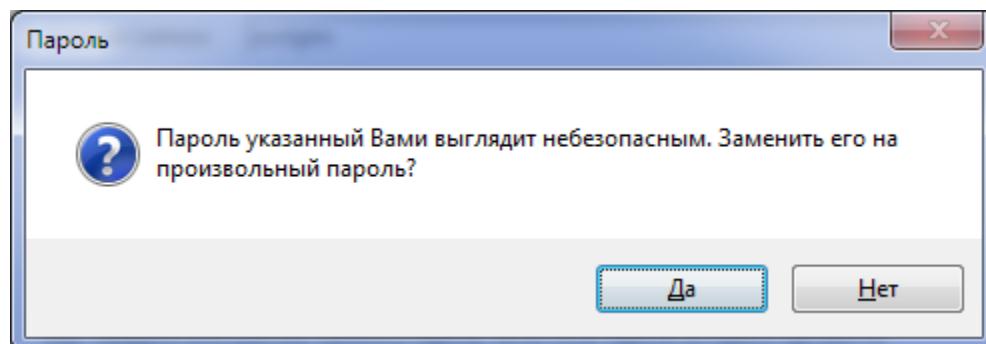


Рисунок 7. Установка Postgres. Шаг 6

10. В следующем окне введите пароль для пользователя `postgres` в базе данных. Рекомендуем использовать пароль `postgres`. Выберите в качестве кодировки сервера и кодировки клиента UTF-8 .

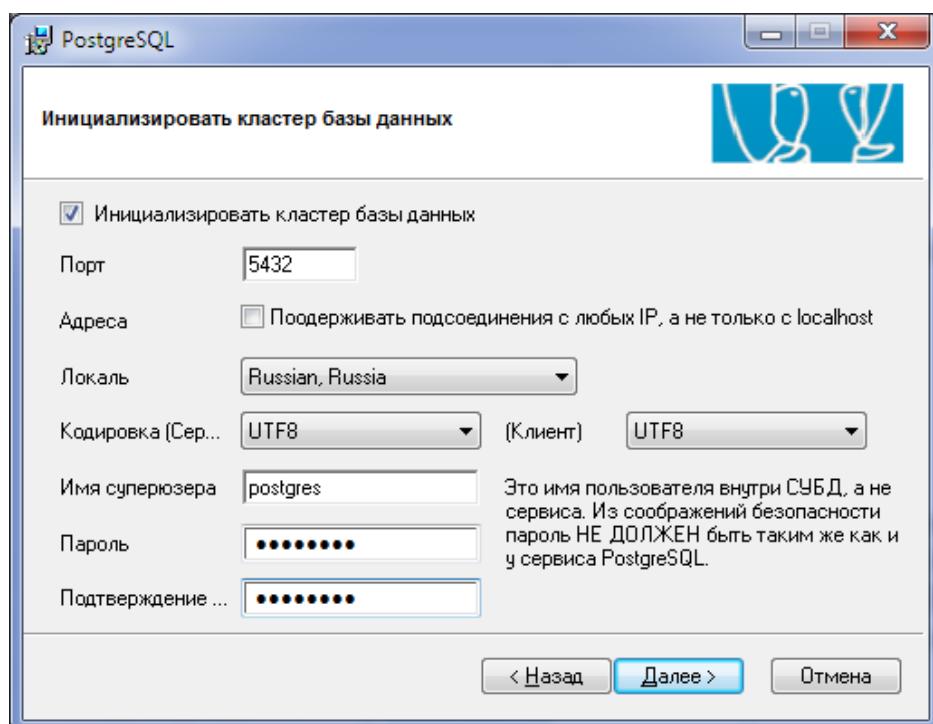


Рисунок 8. Установка Postgres. Шаг 7

11. В следующем окне нажмите на кнопку **Далее**.

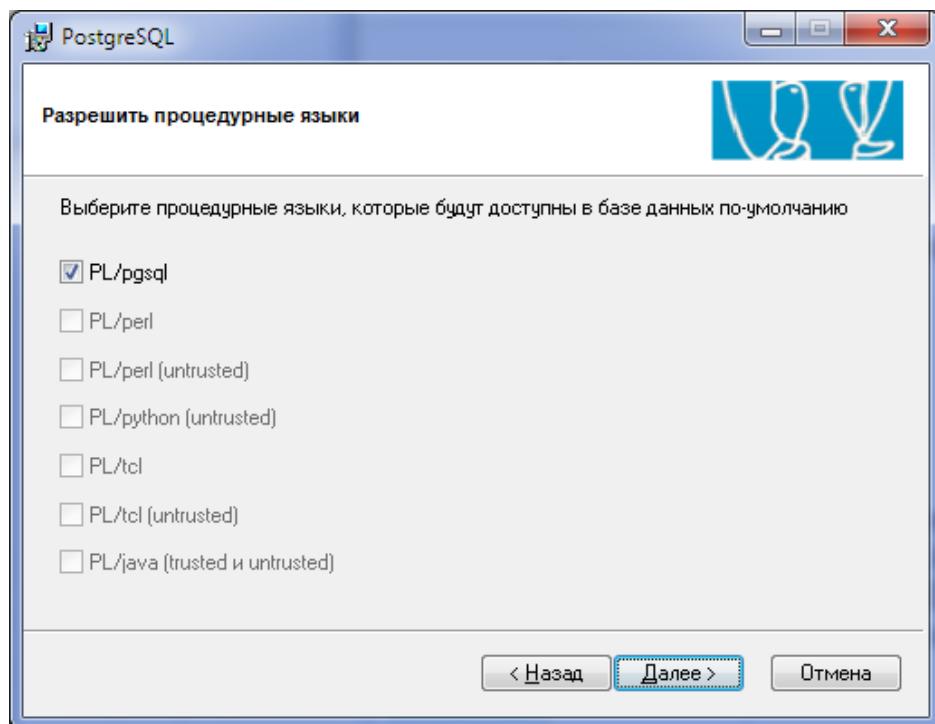
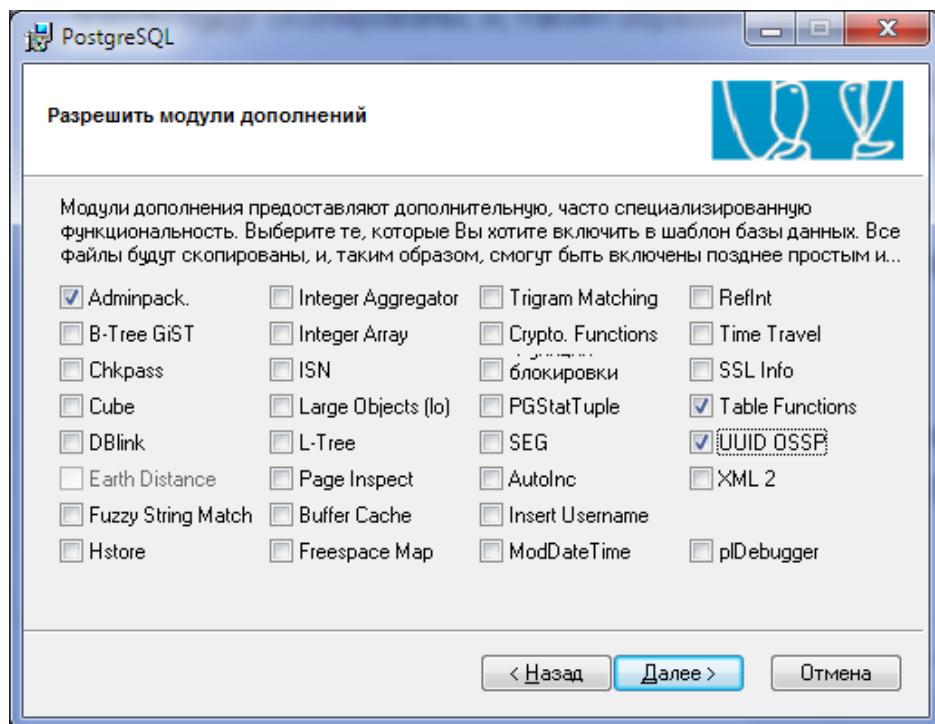


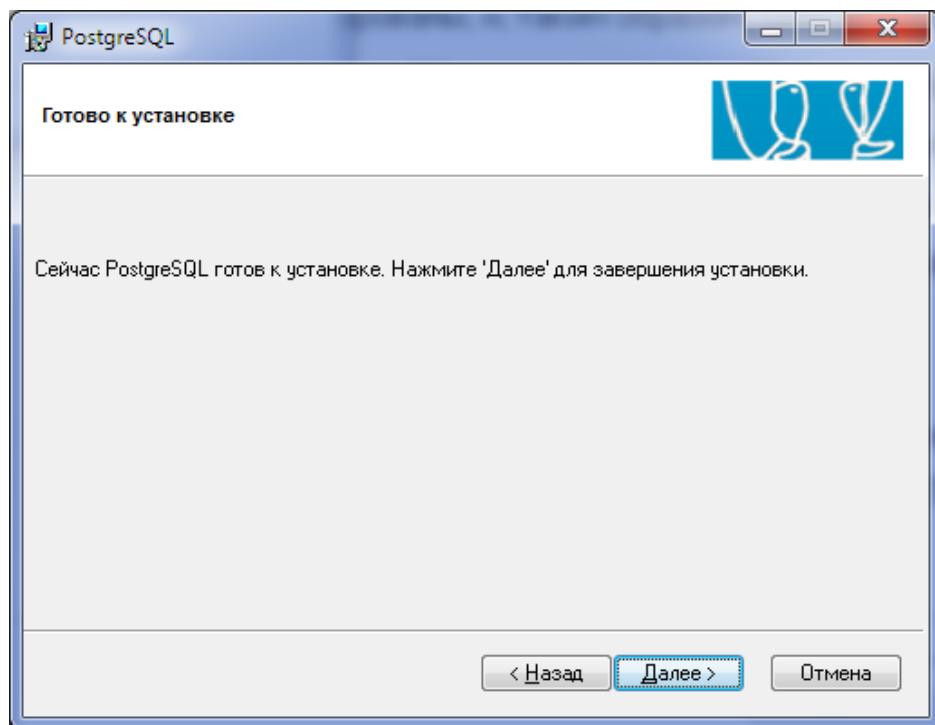
Рисунок 9. Установка Postgres. Шаг 8

12. В следующем окне установите следующие галочки: **Adminpack**, **Table Functions**, **UUID OSSP**.



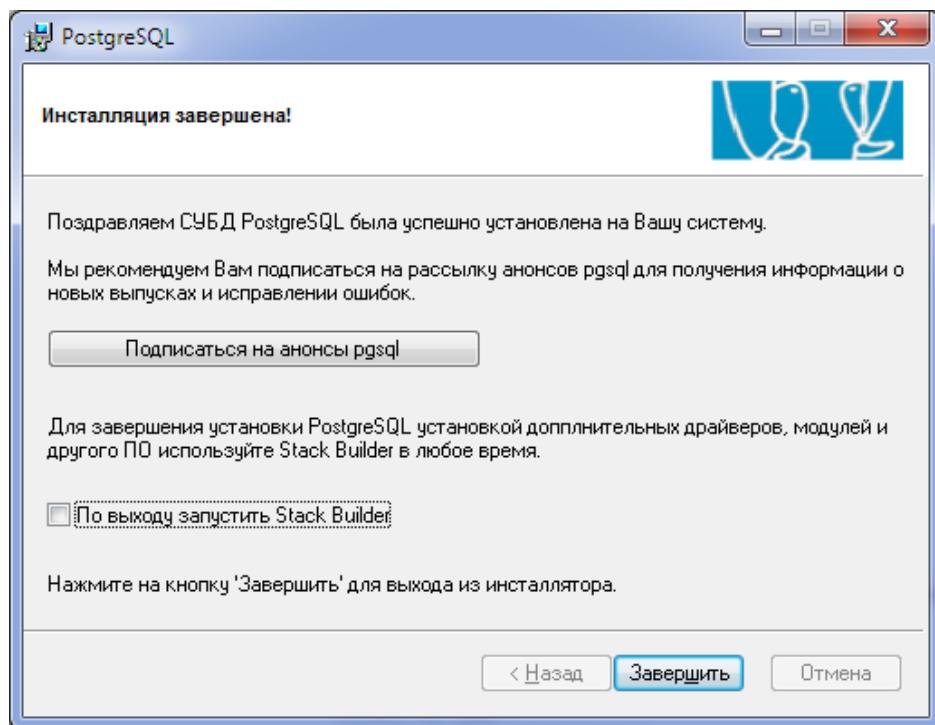
**Рисунок 10. Установка Postgres. Шаг 9**

13. В следующем окне нажмите на кнопку **Далее**.



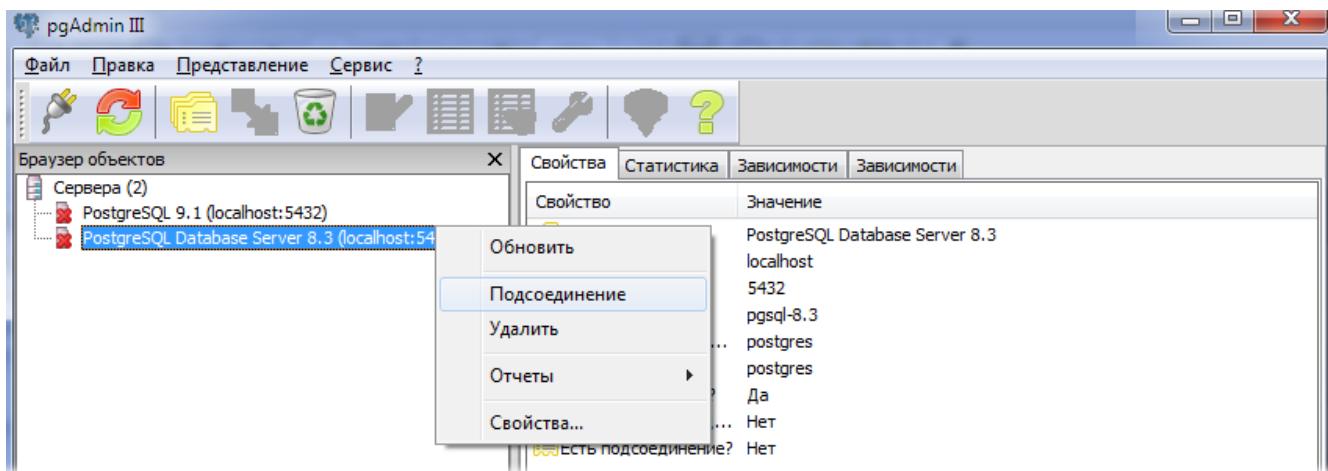
**Рисунок 11. Установка Postgres. Шаг 10**

14. В следующем окне отображается информация об успешном завершении установки. Нажмите на кнопку **Завершить**.



**Рисунок 12. Установка Postgres. Шаг 11**

После установки убедимся в работоспособности сервера базы данных. Для этого зайдите в меню **Пуск**, далее – **PostgreSQL 8.3**, далее – **pgAdmin III**. На экране откроется окно, представленное на рисунке:



**Рисунок 13. Главное окно pgAdmin**

Создадим нового пользователя. Для этого нажмите правой клавишей мыши на

**PostgreSQL Database Server 8.3 (localhost:5432)** и в отобразившемся контекстном меню выберите **Подсоединение**. Далее нажмите правой клавишей мыши на **Роли входа** и в отобразившемся контекстном меню выберите **Новое правило**. На экране отобразится окно, представленное на рисунке:

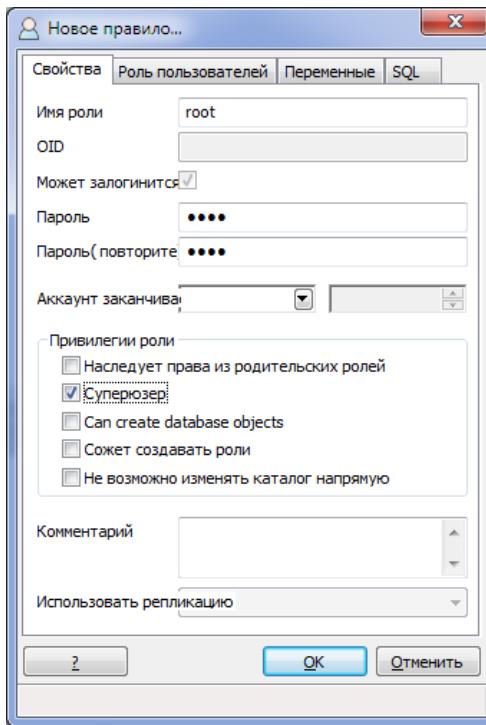


Рисунок 14. Окно создания нового правила

В этом окне в качестве имени роли и пароля укажите `root`. В списке **Привилегии роли** установите флажок **Суперзр**.

Установка и настройка базы данных завершена.

## 1.2. Установка Java SE 6

Сайт компании-разработчика: <http://oracle.com/technetwork/java/>

Дистрибутив: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

1. Скачайте последнюю версию **JDK**, например, `jdk-6u29-windows-x64.exe`.
2. Запустите файл `jdk-6u29-windows-x64.exe`.
3. В первом окне нажмите на кнопку `Next`.

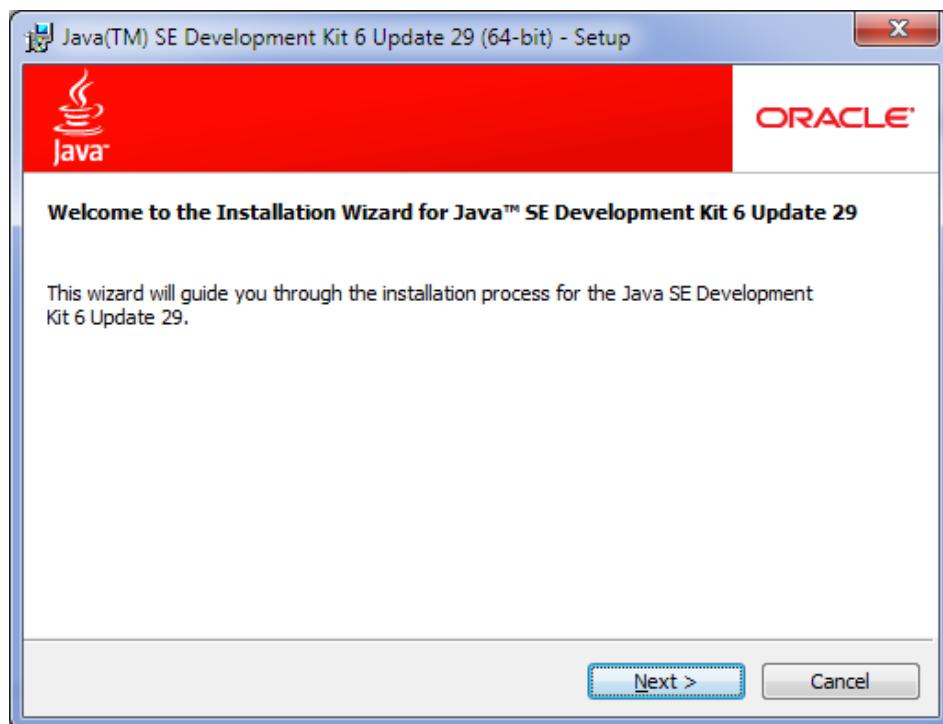
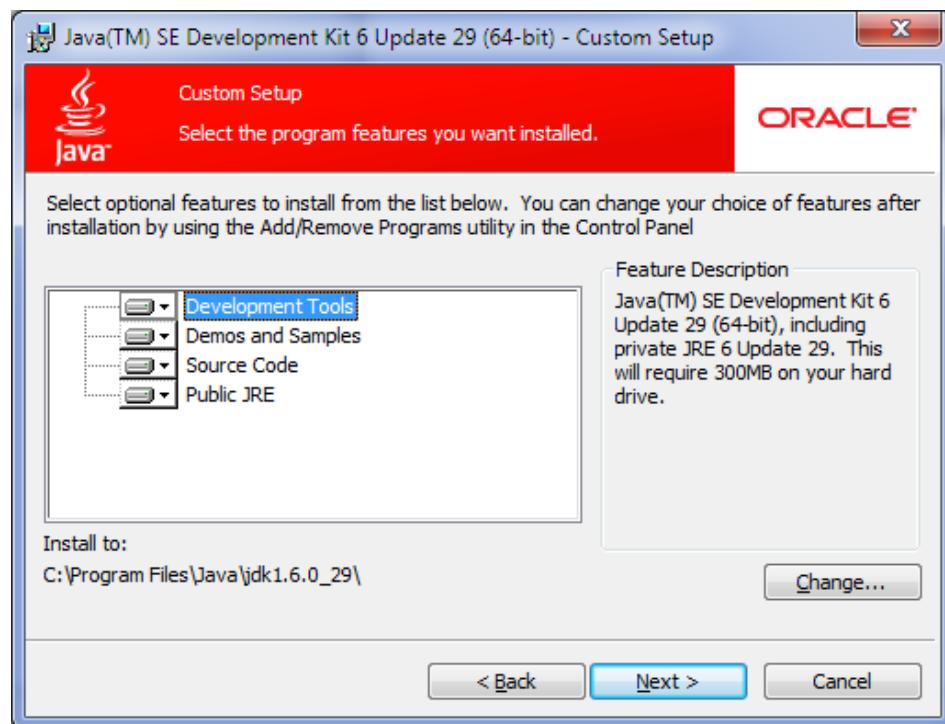


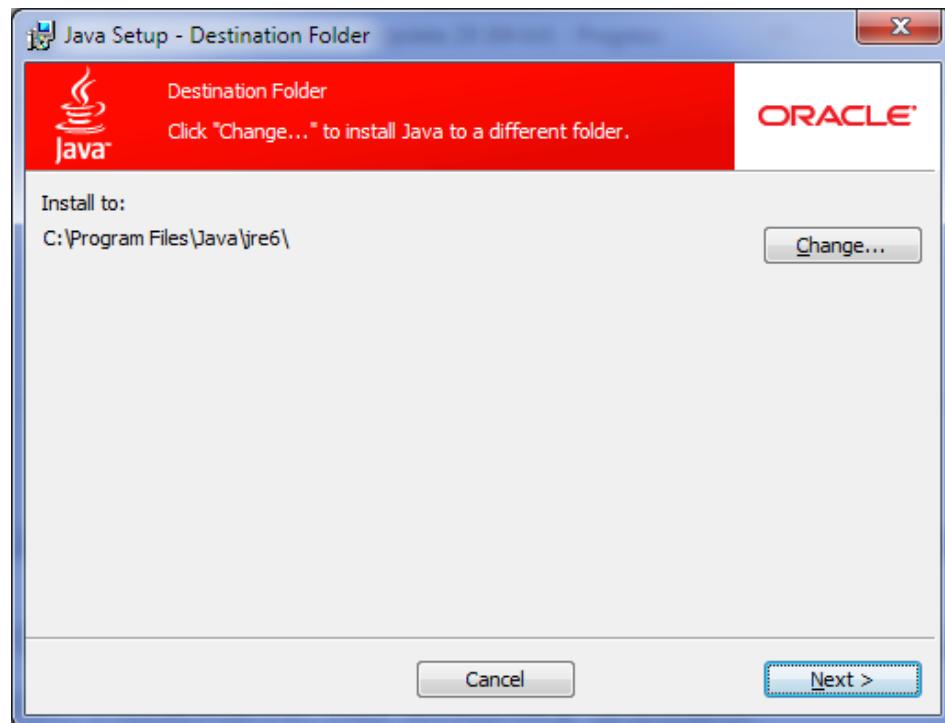
Рисунок 15. Установка Java SE. Шаг 1

4. В следующем окне Вы можете выбрать дополнительные компоненты и сменить каталог установки программы, нажав на кнопку **Change** и выбрав новый путь. Нажмите на кнопку **Next**.



**Рисунок 16. Установка Java SE. Шаг 2**

5. В следующем окне нажмите на кнопку **Next**.



**Рисунок 17. Установка Java SE. Шаг 3**

6. В следующем окне отображается информация об успешном завершении установки. Нажмите на кнопку **Finish**.

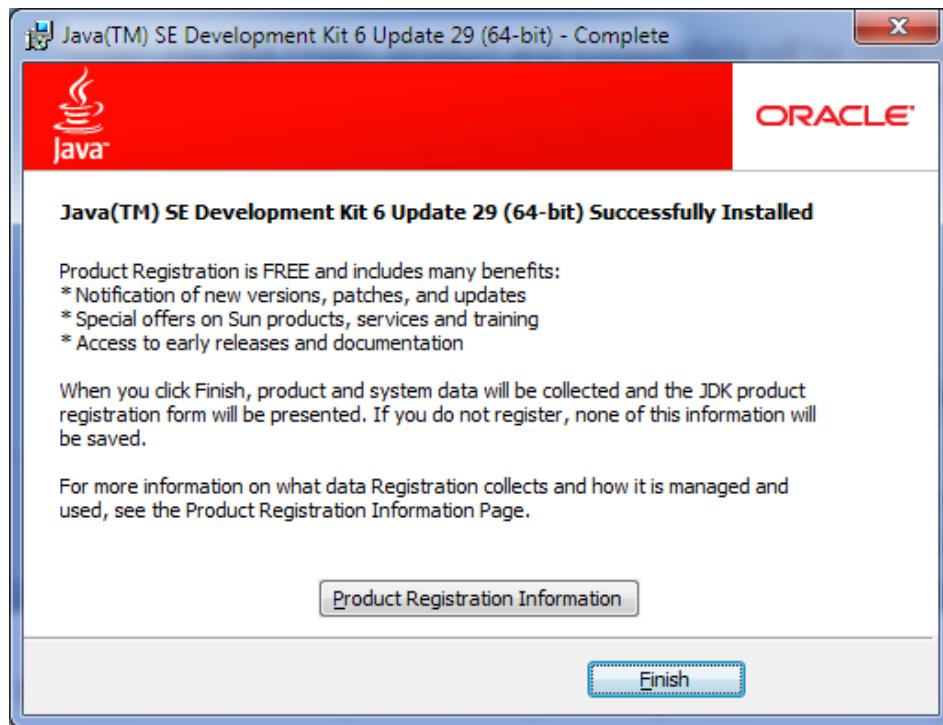


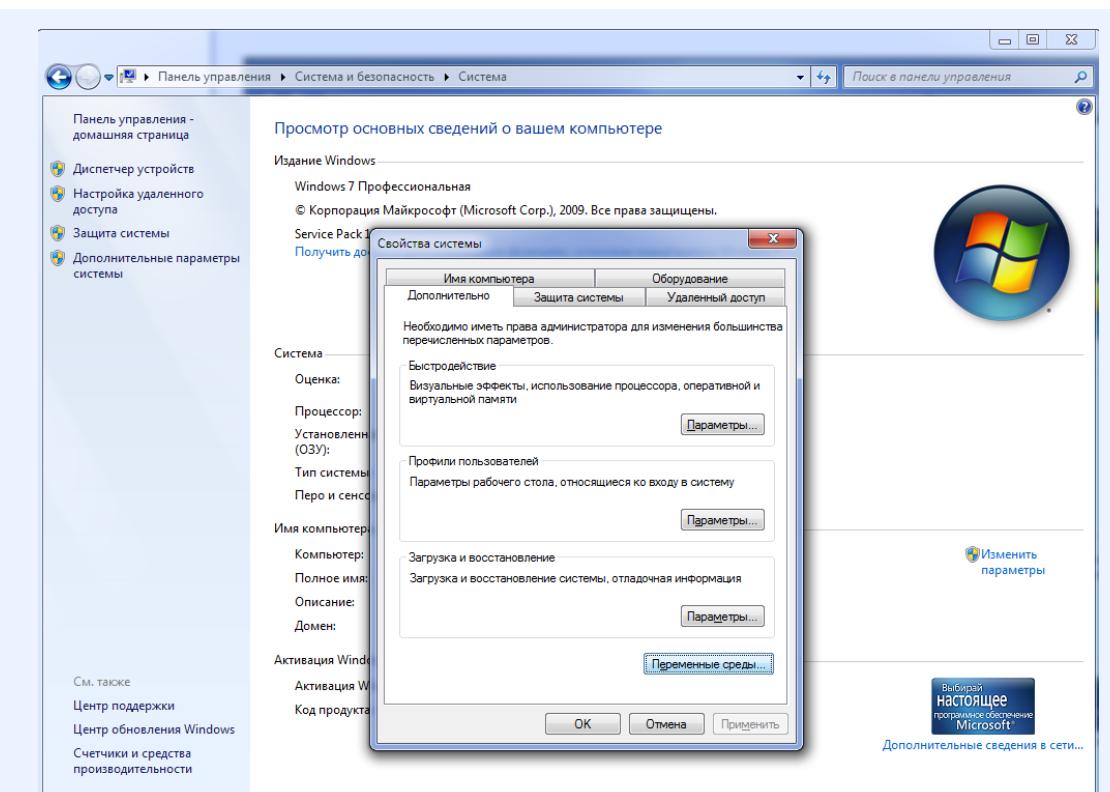
Рисунок 18. Установка Java SE. Шаг 4

7. Теперь создайте переменную среды **JAVA\_HOME**. В качестве значения переменной окружения **JAVA\_HOME** установите путь до каталога установки **JDK**, например, **C:\Program Files\Java\jdk1.6.0\_29**.

#### Подсказка

Как установить или изменить переменную среды?

В пункте меню **Пуск** выберите **Панель управления**, далее в элементах панели управления выберите элемент **Система**. На панели слева найдите ссылку на **Дополнительные параметры системы** и нажмите на нее. В отобразившемся окне **Свойства системы** перейдите на вкладку **Дополнительно** и нажмите на кнопку **Переменные среды**.



**Рисунок 19. Окно изменения свойств системы**

Чтобы создать новую переменную среды, нажмите на кнопку **Создать** в панели **Системные переменные**.

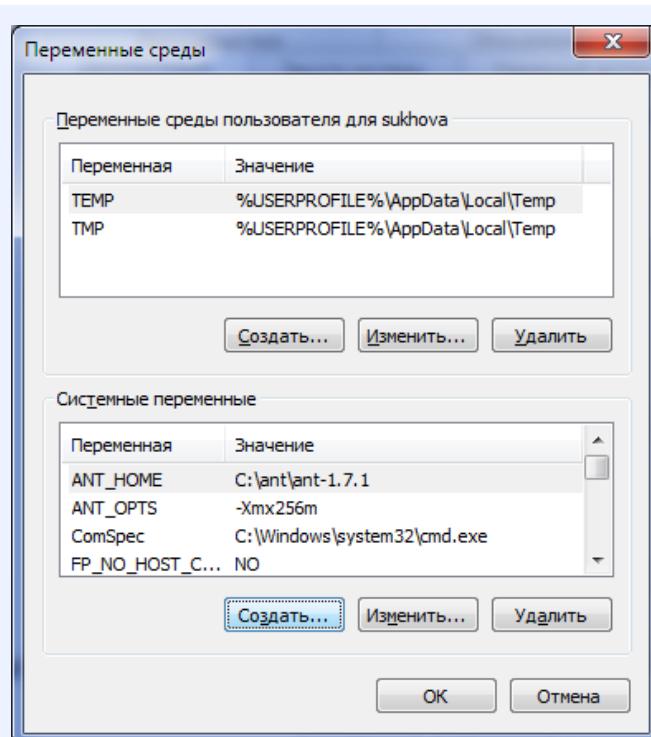


Рисунок 20. Окно управления переменными среды

В поле **Имя переменной** введите **JAVA\_HOME** , а в поле **Значение переменной** **C:\Program Files\Java\jdk1.6.0\_29**.

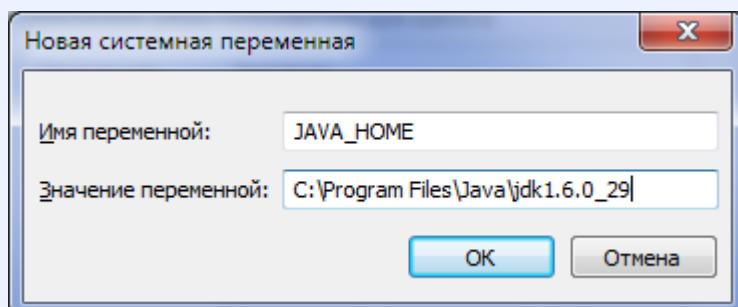


Рисунок 21. Окно создания переменной среды JAVA\_HOME

Чтобы изменить значение существующей переменной среды, например, **Path** , найдите ее в списке на панели **Системные переменные** и нажмите на кнопку **Изменить**. В поле **Значение переменной** введите значение **%JAVA\_HOME%\bin**.

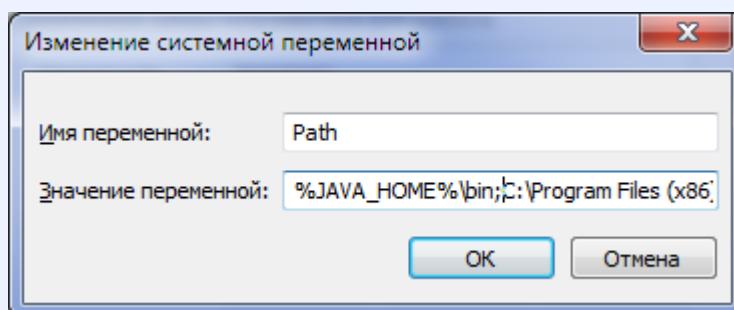


Рисунок 22. Окно изменения переменной среды Path

8. К значению переменной среды `Path` добавьте значение `%JAVA_HOME%\bin`.

Значение переменной `Path` необходимо обновить для того, чтобы исполняемые файлы **JDK** стали доступны в командной строке.

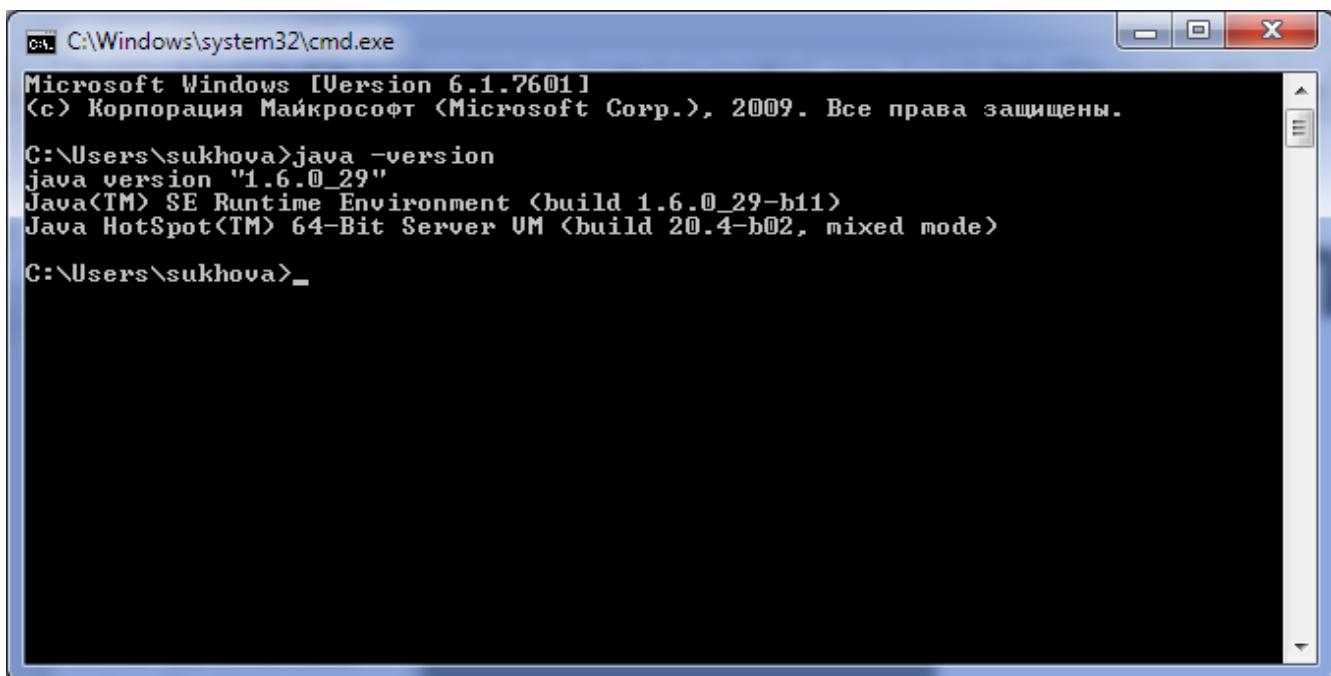
После установки убедимся в работоспособности java-машины. Для этого запустите командную строку и введите в ней команду

```
java -version .
```

#### Совет

Для того чтобы запустить командную строку **Windows**, нажмите сочетание клавиш **WIN +R** и в отобразившемся окне наберите `cmd`.

В результате выполнения команды Вы увидите окно командной строки, вид которого представлен на следующем рисунке:



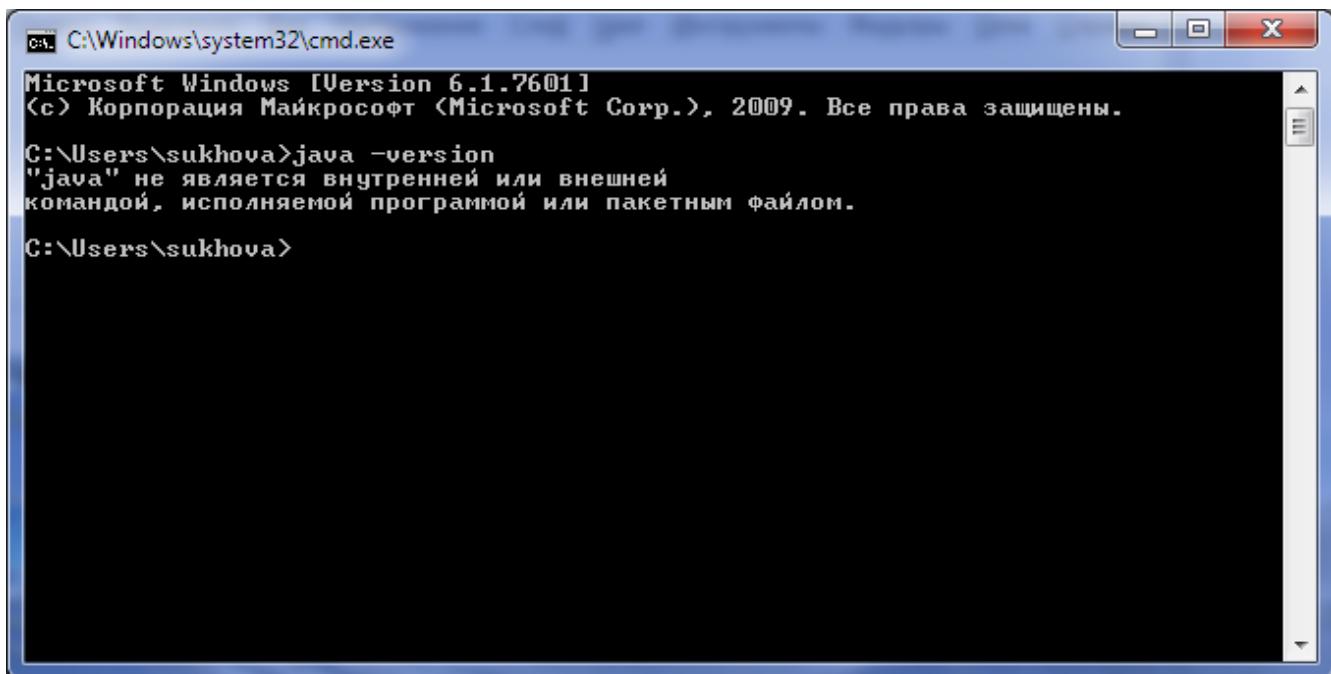
```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Users\sukhova>java -version
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Java HotSpot(TM) 64-Bit Server VM (build 20.4-b02, mixed mode)

C:\Users\sukhova>
```

Рисунок 23. Окно командной строки. Результат команды `java -version`

Если в результате выполнения команды Вы увидите окно, вид которого представлен ниже, вернитесь к пункту 7 и проверьте правильность установки переменных среды `JAVA_HOME` и `Path`.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Users\sukhova>java -version
"java" не является внутренней или внешней
командой, исполняемой программой или пакетным файлом.

C:\Users\sukhova>
```

Рисунок 24. Окно командной строки. Результат команды `java -version`

Установка и настройка **Java** завершена.

## 1.3. Установка Gradle 1.0-milestone-3

Сайт компании-разработчика: <http://www.gradle.org/>

Дистрибутив: <http://repo.gradle.org/gradle/distributions/gradle-1.0-milestone-3-all.zip>

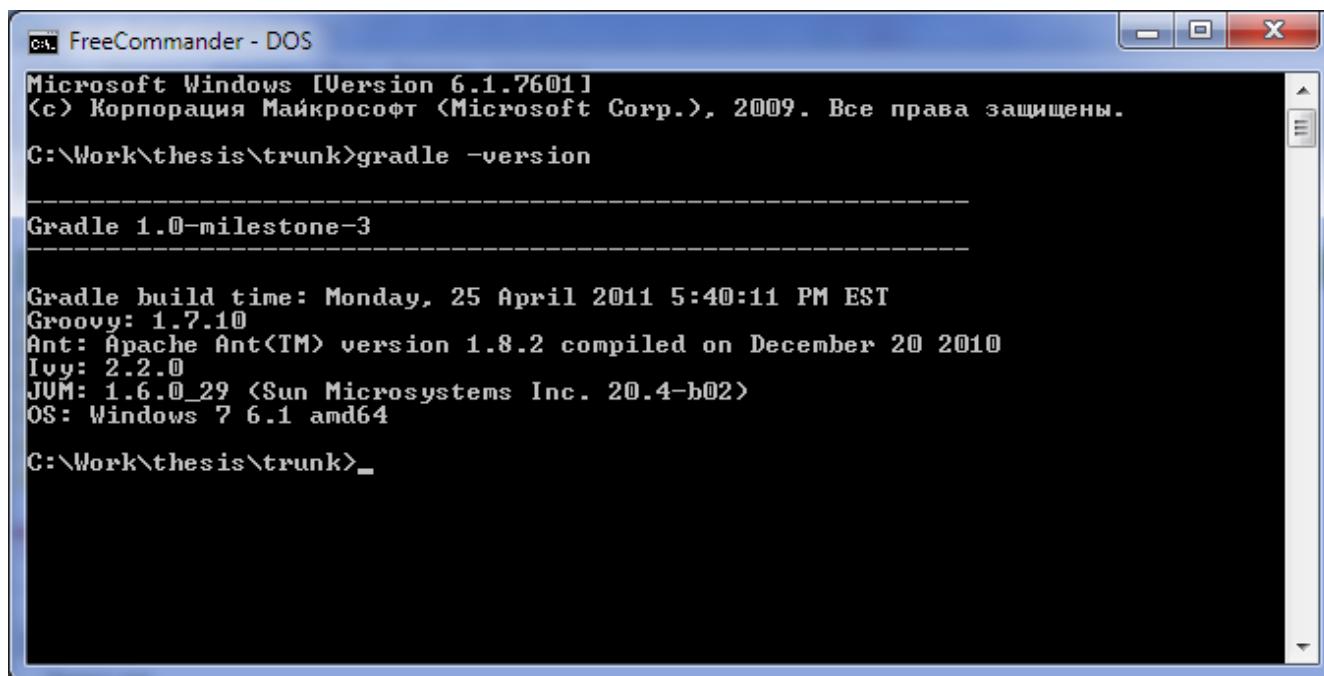
1. Скачайте архив gradle-1.0-milestone-3-all.zip и распакуйте его, например, в C:\Program Files (x86)\gradle .
2. К значению переменной среды Path добавьте значение

C:\Program Files (x86)\gradle\gradle-1.0-milestone-3\bin.

Убедимся в работоспособности **gradle**. Для этого в командной строке введите команду

gradle -version

В результате выполнения команды Вы увидите окно командной строки, вид которого представлен на следующем рисунке:



```
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

C:\Work\thesis\trunk>gradle -version

-----
Gradle 1.0-milestone-3
-----

Gradle build time: Monday, 25 April 2011 5:40:11 PM EST
Groovy: 1.7.10
Ant: Apache Ant(TM) version 1.8.2 compiled on December 20 2010
Ivy: 2.2.0
JVM: 1.6.0_29 (Sun Microsystems Inc. 20.4-b02)
OS: Windows 7 6.1 amd64

C:\Work\thesis\trunk>
```

Рисунок 25. Окно командной строки. Результат команды gradle -version

## 1.4. Установка IntelliJ IDEA 11 Community Edition

Сайт компании-разработчика: <http://www.jetbrains.com/>

Дистрибутив: <http://download-1n.jetbrains.com/idea/ideaIC-11.exe>

1. Скачайте последнюю версию **IntelliJ IDEA Community Edition**. На момент написания данного руководства это **IntelliJ IDEA 11**.
2. Запустите файл `ideaIC-11.exe`.
3. В первом окне нажмите на кнопку **Next**.

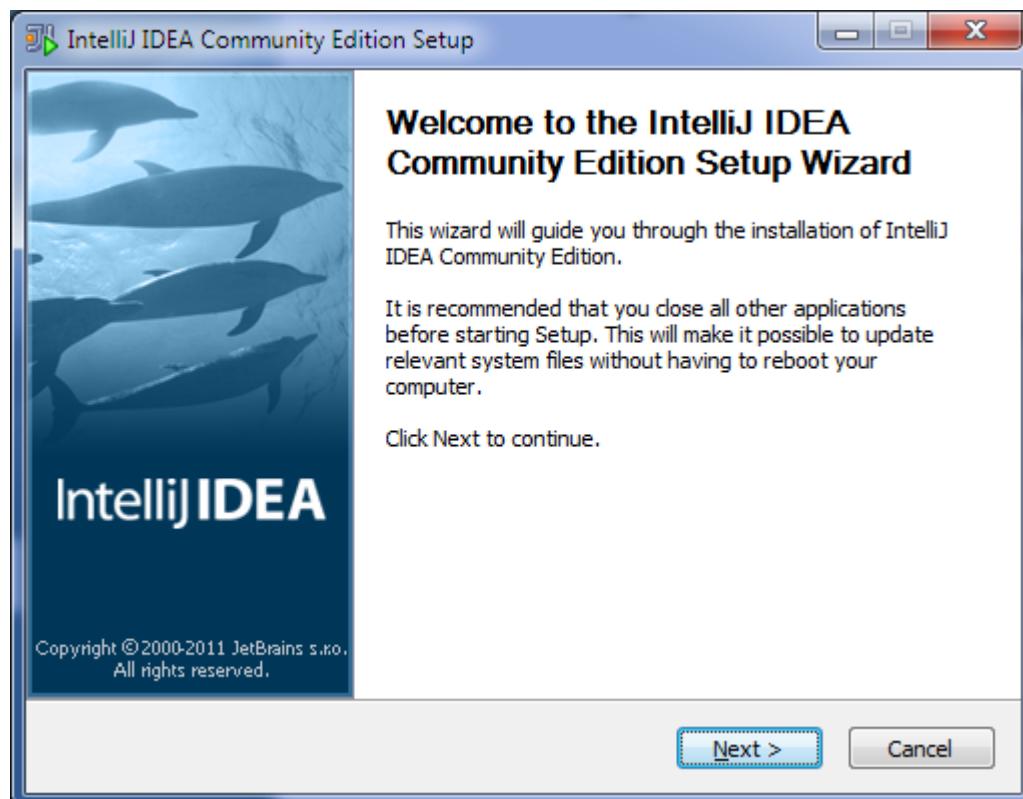


Рисунок 26. Установка IntelliJ IDEA. Шаг 1

4. В следующем окне предоставляется возможность сменить каталог установки программы, нажав на кнопку **Browse**.

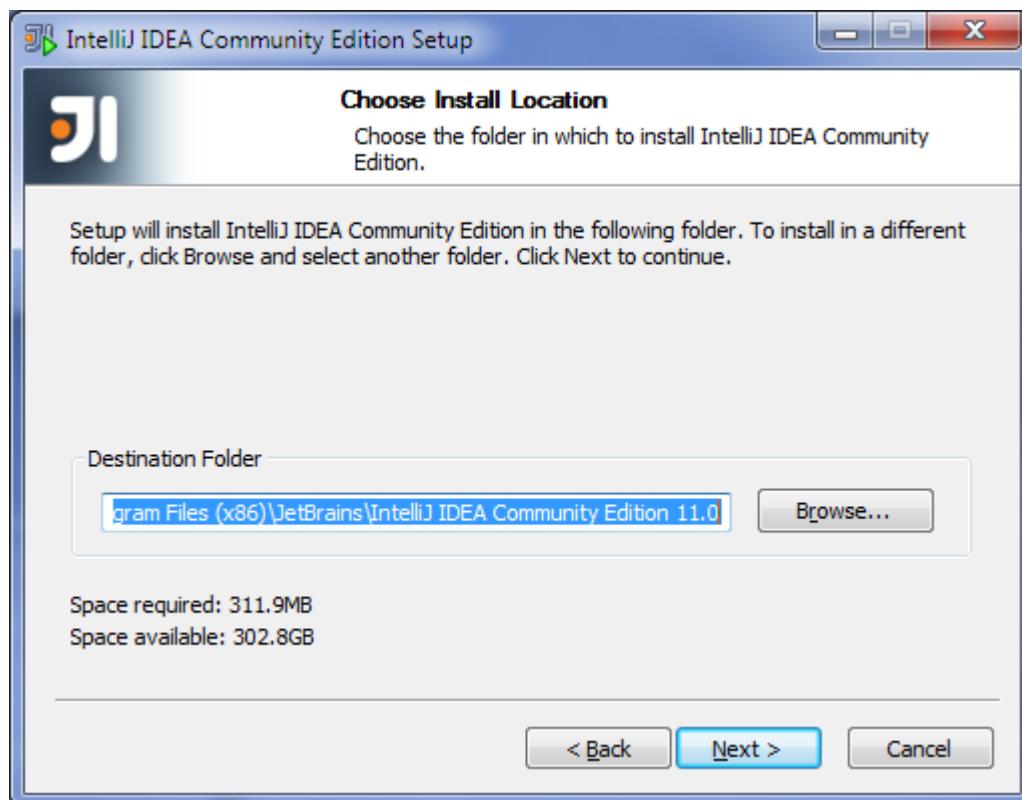


Рисунок 27. Установка IntelliJ IDEA. Шаг 2

5. В следующем окне предоставляется возможность установления ярлыка на рабочем столе и ярлыка быстрого запуска приложения.

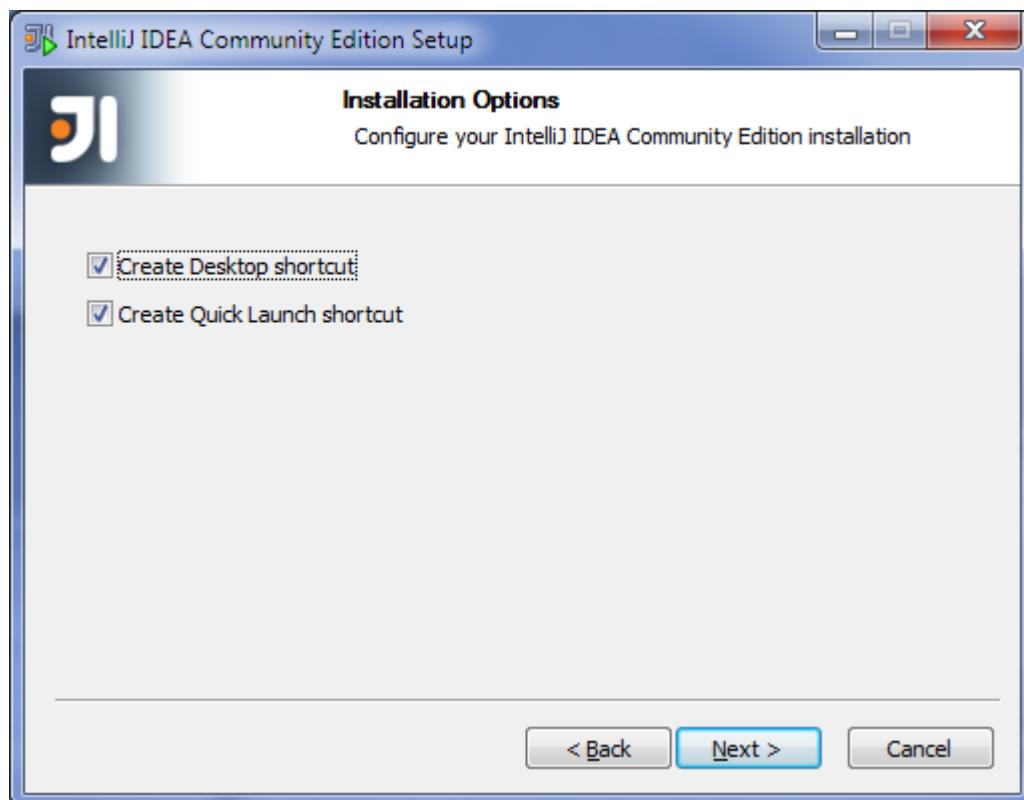


Рисунок 28. Установка IntelliJ IDEA. Шаг 3

6. Далее можно задать имя папки, которое будет отображаться в меню Пуск .

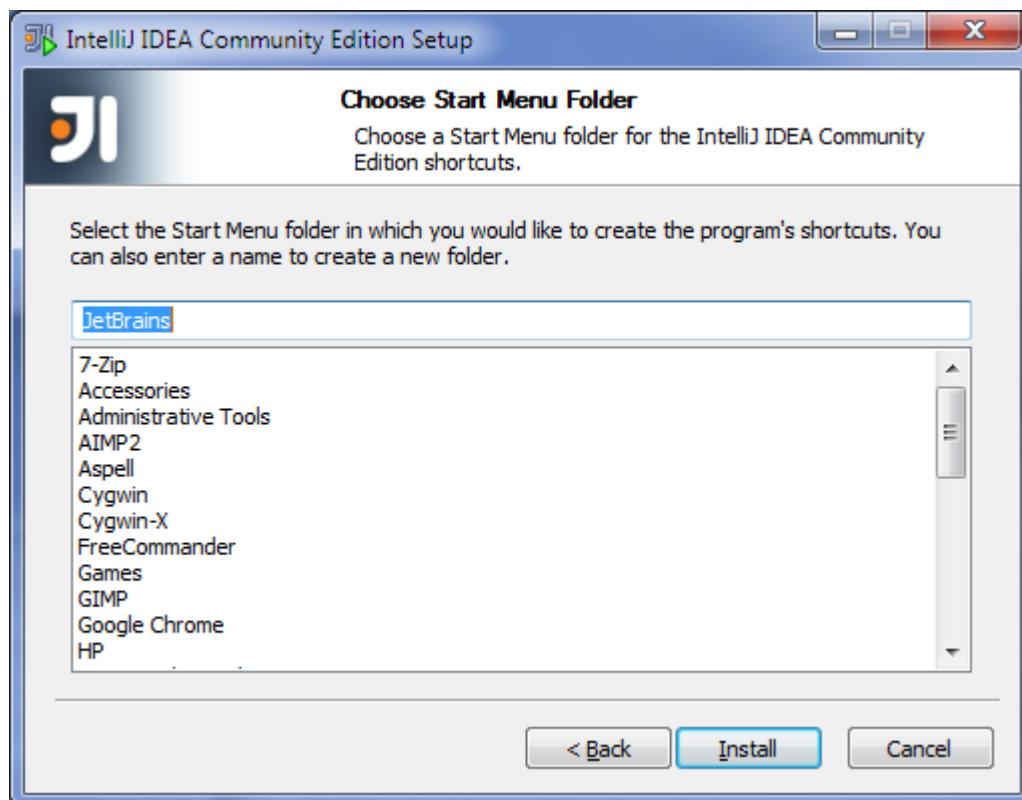


Рисунок 29. Установка IntelliJ IDEA. Шаг 4

7. В следующем окне установите галочку **Run IntelliJ IDEA Community Edition**.

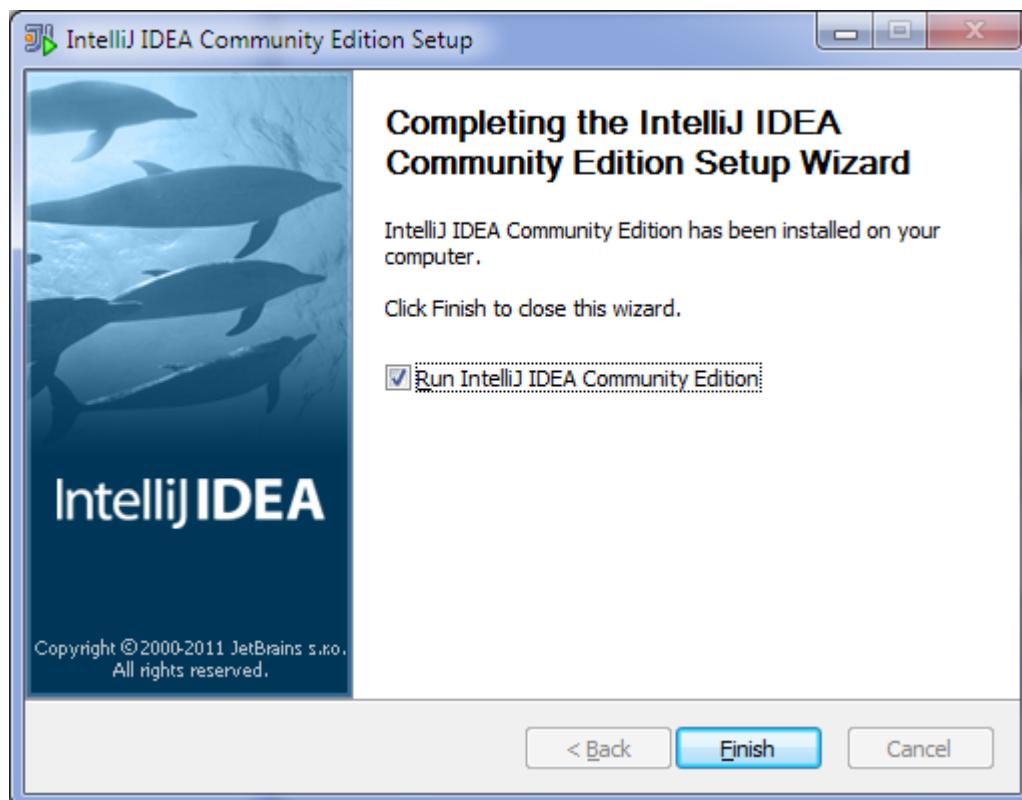


Рисунок 30. Установка IntelliJ IDEA. Шаг 5

- Произведем настройку среды разработки. В отобразившемся окне предлагается импортировать настройки, если другая версия **IDEA** уже была установлена на данном локальном компьютере. Если вы устанавливаете **IDEA** в первый раз, установите переключатель в положение **I do not have a previous version of IntelliJ IDEA or I do not want to import my settings** и нажмите на кнопку **OK**.

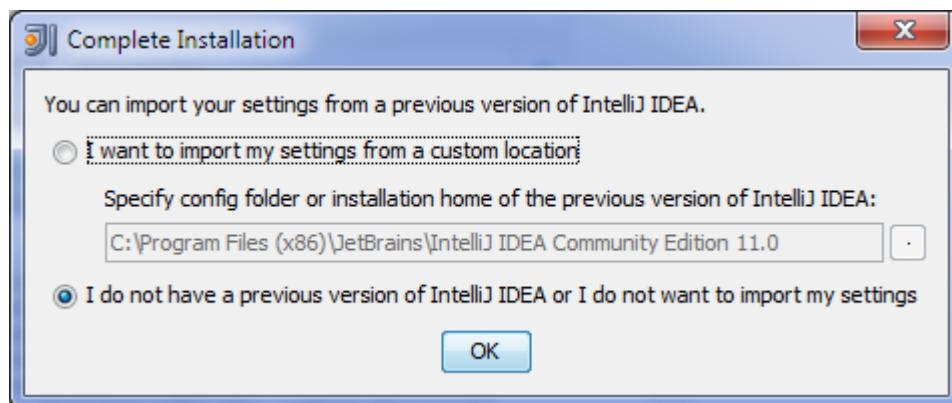
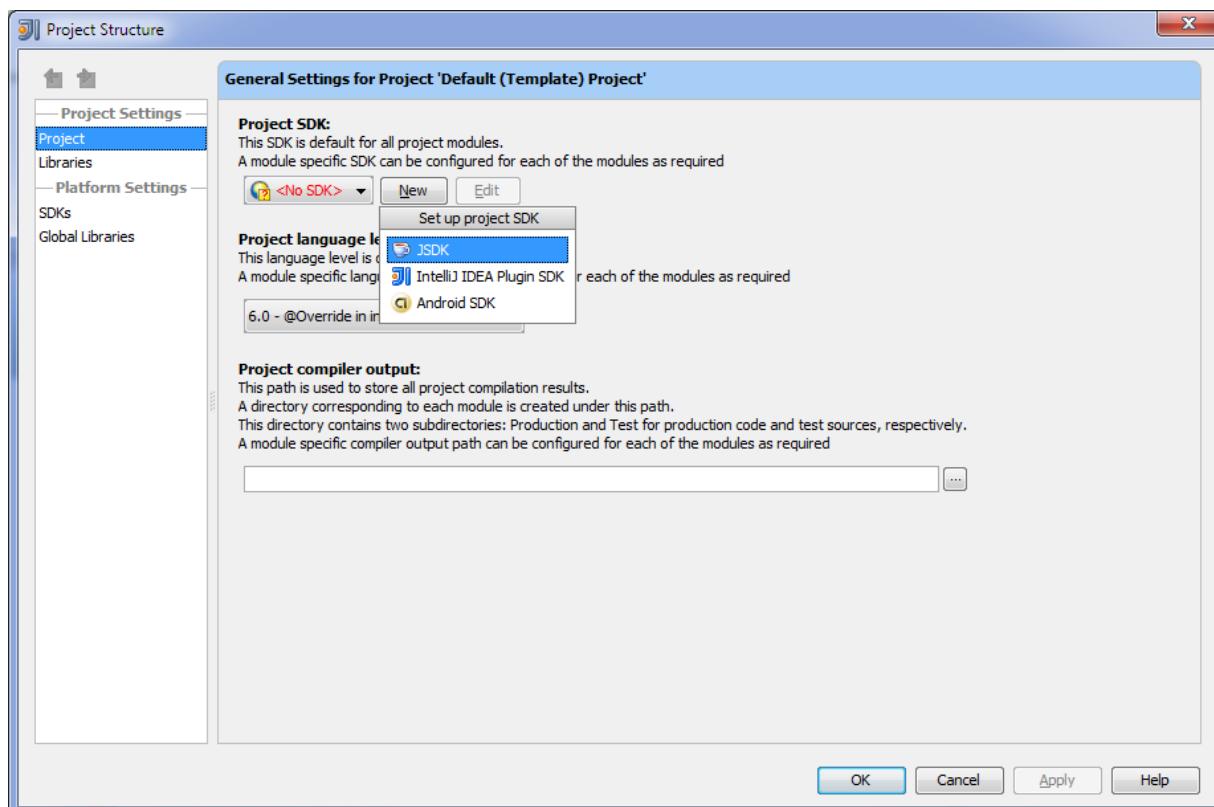


Рисунок 31. Установка IntelliJ IDEA. Шаг 6

9. Теперь установим **SDK**, которое будет использоваться по умолчанию. Для этого в меню **File** выберите пункт меню **Project Structure**. Далее слева в группе **Project Settings** выберите строку **Project**. На отобразившейся панели настроек нажмите на кнопку **New** и выберите в контекстном меню пункт **JSDK**.



**Рисунок 32. Установка IntelliJ IDEA. Шаг 7**

В отобразившемся окне укажите путь к папке с **JDK**, например, `C:\Program Files\Java\jdk1.6.0_29`. Для сохранения настроек нажмите на кнопку **Apply**.

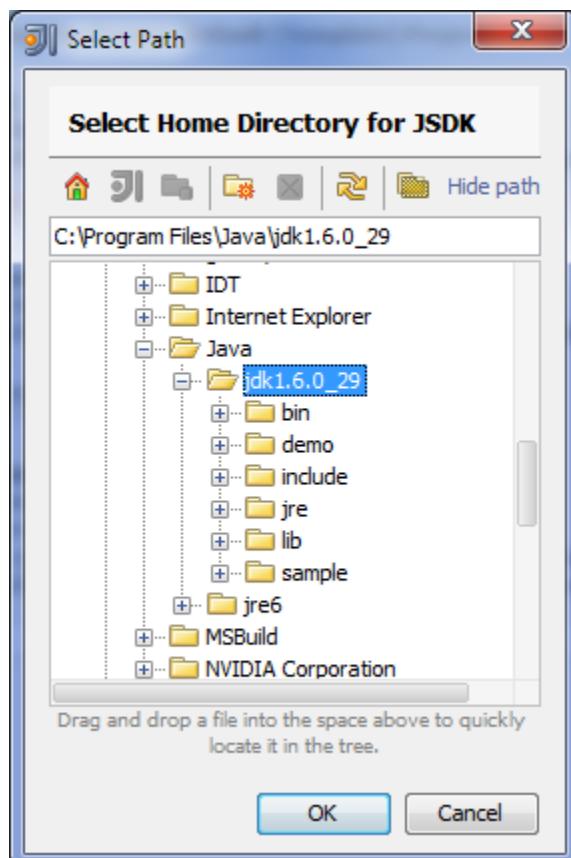


Рисунок 33. Установка IntelliJ IDEA. Шаг 8

Установка и настройка среды разработки **IntelliJ IDEA** завершена.

## Глава 2. Создание и запуск проекта расширения для системы ТЕЗИС

Для создания проекта расширения выполните следующие действия:

1. Создайте рабочую папку для скрипта-сборщика, например, `c:/work/thesis-builder`
2. Создайте в этой папке файл `build.gradle` следующего содержания:

```
/*
 * Usage:
 * 'gradle thesis' - builds Thesis distribution in 'build/target/thesis'
 * 'gradle distr' - builds and packs Thesis distribution in 'build/distributions'
 * 'gradle ext' - builds a new extension project in 'build/target/ext'
 *
 * The set of addons must be defined below in 'buildDefinition' section
 */

// Support for loading Thesis plugin from the repository
buildscript {
    org.apache.ivy.util.url.CredentialsStore.INSTANCE.addCredentials(
        'Sonatype Nexus Repository Manager',
        'repository.haulmont.com',
        System.getenv('HAULMONT_REPOSITORY_USER'),
        System.getenv('HAULMONT_REPOSITORY_PASSWORD')
    )
    repositories {
        mavenLocal()
        mavenRepo(urls:
            "http://repository.haulmont.com:8587/nexus/content/groups/work")
    }
    dependencies {
        classpath group: 'com.haulmont.thesis.gradle',
            name: 'thesis-plugin',
            version: '1.0-SNAPSHOT'
    }
}

// Required property: platform artifacts version
platformVersion = '3.0-SNAPSHOT'

// Required property: Thesis artifacts version
thesisVersion = '3.2-SNAPSHOT'

reminderVersion = '1.3-SNAPSHOT'

// Optional property: name for extension project ('ext' by default)
extensionName = 'myext'

apply(plugin: 'thesis')

// Define build contents - order of components is important
buildDefinition {
    // Add standard Thesis (optional, it is added anyway)
    mainApp()
```

```
// Add 'Requests' addon
// requestsAddon()
}
```

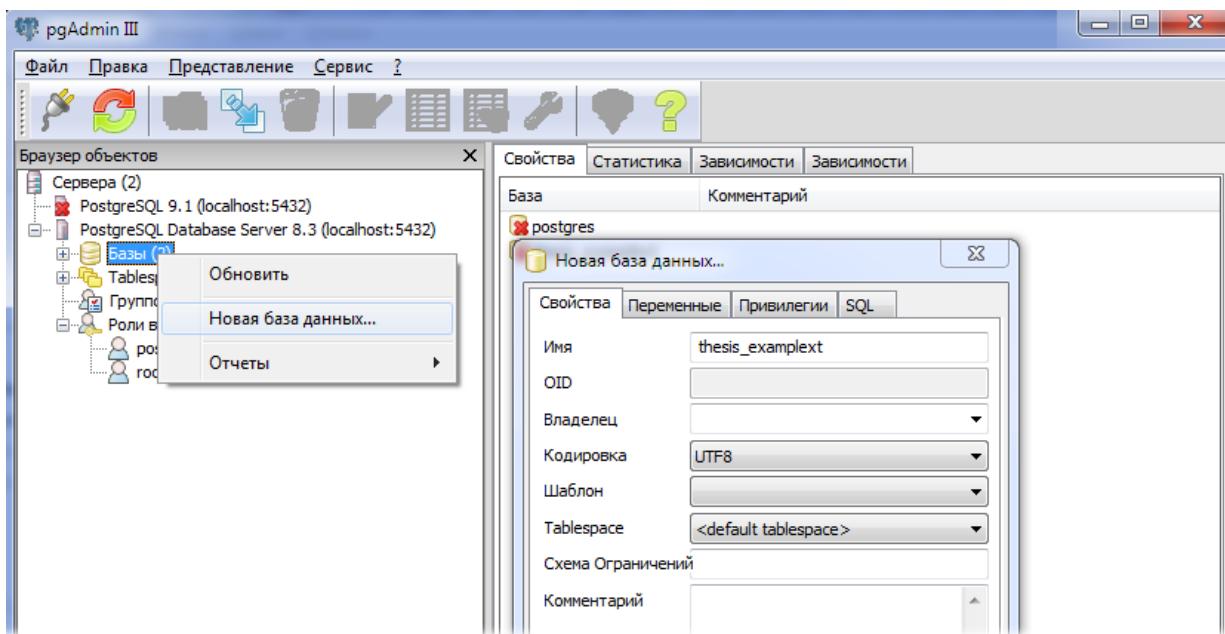
3. Задайте имя создаваемого расширения в строке `extensionName` . Оно должно быть как можно более простым и коротким и содержать только буквы латинского алфавита в нижнем регистре.
4. Запустите командную строку в каталоге `c:/work/thesis-builder` .
5. Теперь можно создать шаблон проекта с помощью команды

`gradle ext`

В результате выполнения команды в подкаталоге `target/ext` будет создана заготовка типового проекта со всеми основными файлами и структурой каталогов.

6. Далее скопируйте получившуюся заготовку проекта в новую рабочую папку, например `c:/work/thesis-ext` , в которой и будет производится разработка расширения.
7. Далее создадим пустую базу данных с названием `thesis_имя_расширения` . Для этого откройте `pgAdmin` : зайдите в меню **Пуск** , далее – **PostgreSQL 8.3** , далее выберите **pgAdmin III** .

В отобразившемся окне нажмите правой клавишей мыши на **PostgreSQL Database Server 8.3 (localhost:5432)** и в отобразившемся контекстном меню выберите **Подсоединение** . Далее нажмите правой клавишей мыши на **Базы** и в отобразившемся контекстном меню выберите **Новая база данных** . На экране отобразится окно, представленное на рисунке:



**Рисунок 34. Создание новой базы данных**

В качестве имени укажите **thesis\_имя\_расширения** . Далее нажмите кнопку **OK**. Новая база данных создана.

8. Собираем проект. На этом этапе будут загружены все необходимые библиотеки и в каталогах модулей в подкаталогах `build` будут собраны артефакты проекта.

В командной строке введите команду

```
gradle assemble
```

9. Устанавливаем сервер приложений **Tomcat** :

```
gradle setupTomcat
```

10. Разворачиваем артефакты и библиотеки проекта в **Tomcat** :

```
gradle deploy
```

11. Запускаем сервер:

```
gradle start
```

12. Чтобы работать с проектом в **IntelliJ IDEA** , запускаем команду, с помощью которой собираются проектные файлы `*.ipr` , `*.iml` .

```
gradle idea
```

13. Для просмотра приложения в веб-браузере введите в адресную строку

<http://localhost:8080/app>

Имя учетной записи и пароль – admin/admin.

## 2.1. Структура и назначение каталогов и файлов проекта

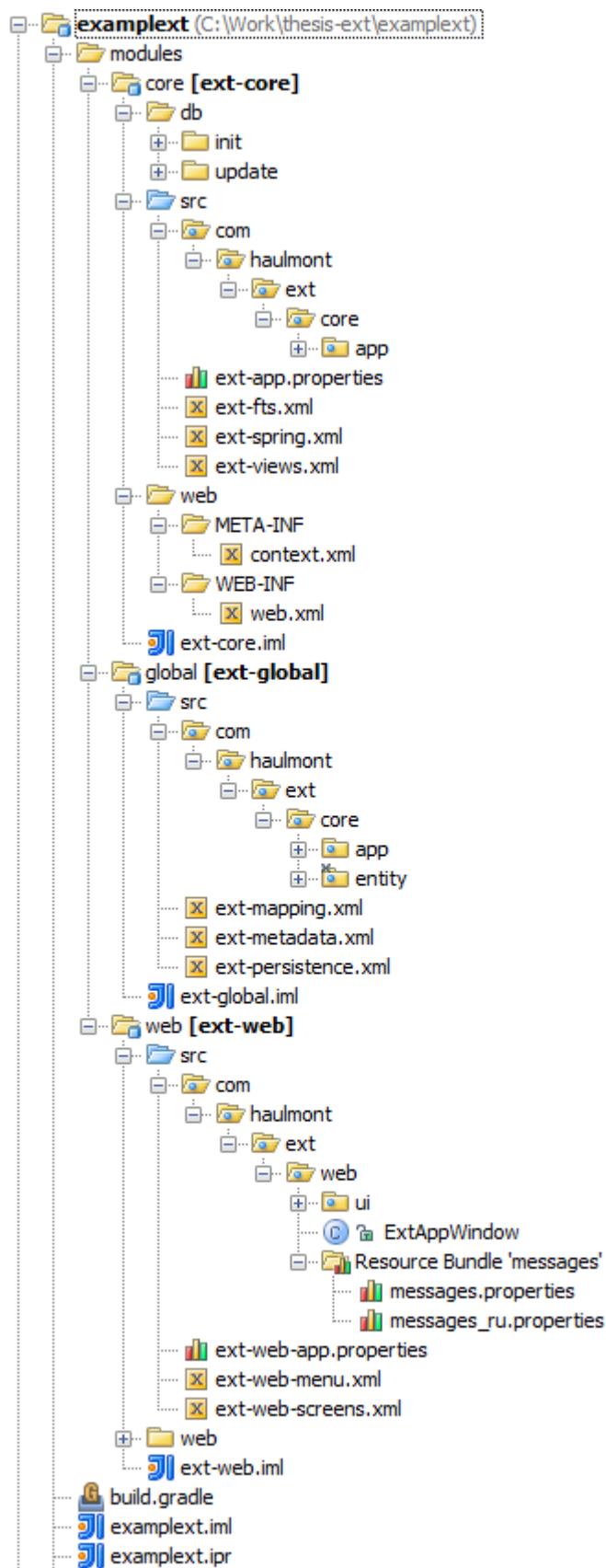
Рассмотрим структуру классов и других файлов в проекте.

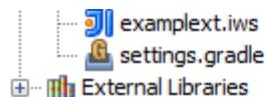
В корне проекта расположены скрипты сборки `build.gradle`, `settings.gradle` и проектные файлы **IntelliJ IDEA**.

В каталоге `modules` расположены подкаталоги модулей проекта – `global`, `core`, `web`.

Модуль `global` определяет компоненты, доступные как среднему слою, так и клиентскому коду.

- В пакете `com.haulmont.ext.core.app` создаются интерфейсы сервисов *middleware*.
- В пакете `com.haulmont.ext.core.entity` создаются классы сущностей и файлы локализованных сообщений `messages.properties` для задания названий сущностей и их атрибутов.
- `ext-persistence.xml` – в этом файле регистрируются новые классы сущностей.
- `ext-mapping.xml` – здесь можно переопределить свойства стандартных сущностей.
- `ext-metadata.xml` – в этом файле может быть задано замещение стандартных сущностей новыми.





**Рисунок 35. Структура каталогов и файлов проекта**

Модуль core определяет компоненты среднего слоя системы.

- Каталог db содержит sql-скрипты создания и изменения базы данных в подкаталогах init и update соответственно.
- В пакете com.haulmont.ext.core.app создаются классы реализации сервисов *middleware*, а также другие компоненты бизнес-логики приложения.
- Файл ext-app.properties содержит конфигурационные параметры среднего слоя:

**cuba.springContextConfig** – определяет набор конфигурационных файлов Spring -фреймворка;

**cuba.persistenceConfig** – определяет набор конфигурационных файлов ORM;

**cuba.viewsConfig** – задает набор конфигурационных файлов представлений ;

**cuba.ftsConfig** – задает набор дескрипторов Full Text Search (полнотекстового поиска).

- В файле ext-fts.xml (дескриптор FTS ) можно описать сущности расширения, подлежащие индексированию и поиску .
- В файле ext-spring.xml при необходимости можно подменить реализацию стандартных сервисов или других управляемых фреймворком Spring компонентов, а также зарегистрировать создаваемые JMX-компоненты.
- Файл ext-views.xml содержит описание представлений , используемых в расширении. Механизм представлений обеспечивает извлечение из БД и передачу клиенту графов сущностей, ограниченных в глубину и/или по атрибутам.
- Файл web\META-INF\context.xml содержит настройки подключения к базе данных.
- Файл web\WEB-INF\web.xml – это дескриптор веб-приложения среднего слоя.

Среди прочего содержит список конфигурационных файлов `*-app.properties`, используемых данным приложением.

Модуль `web` определяет компоненты веб-клиента. Это классы *контроллеров*, *XML-дескрипторы* экранов, файлы *локализованных сообщений*.

- В пакете `com.haulmont.ext.web.ui` создаются экраны расширения.
- Класс `com.haulmont.ext.web.ExtAppWindow.java` определяет главное окно приложения.
- Файл `ext-web-app.properties` содержит конфигурационные параметры веб-клиента:

`cuba.windowConfig` – в этом свойстве указывается набор файлов конфигурации экранов приложения;

`cuba.menuConfig` содержит список файлов, описывающих меню приложения; `cuba.mainMessagePack` задает список имен *пакетов сообщений*, формирующих *главный пакет*, используемый главным меню и другими общими компонентами системы;

`thesis.web.appWindow` – имя класса главного окна приложения.

- Файл `ext-web-menu.xml` содержит описание пунктов главного меню расширения.
- В файле `ext-web-screens.xml` регистрируются экраны расширения.

## Глава 3. Быстрый старт

В данном разделе Вы научитесь:

- Создавать скрипт таблицы в базе данных
- Создавать класс сущности
- Создавать экран со списком сущностей
- Создавать экран редактирования сущности
- Расширять существующую функциональность системы

Перед тем как приступить к изучению данного раздела, рекомендуем ознакомиться с разделом Создание и запуск проекта расширения для системы **ТЕЗИС**.

Если у Вас появились вопросы, Вы можете обратиться к разделу Часто задаваемые вопросы.

### 3.1. Создание таблицы в базе данных

Таблицы, относящиеся к разрабатываемому расширению, должны начинаться с префикса EXT. Создадим таблицу **Должность**, имеющую три поля: "**Наименование**", "**Код**" и "**Должностные обязанности**". Скрипт создания таблицы будет выглядеть следующим образом:

```
create table EXT_POST (
    ID uuid,                      --Первичный ключ (системное поле)
    NAME varchar(100),              --Наименование
    CODE varchar(100),              --Код
    JOB_DUTIES varchar(4000),       --Должностные обязанности
    CREATE_TS timestamp,            --Когда создано (системное поле)
    CREATED_BY varchar(50),         --Кем создано (системное поле)
    VERSION integer,                --Версия (системное поле)
    UPDATE_TS timestamp,            --Когда было последнее изменение (системное поле)
    UPDATED_BY varchar(50),          --Кто последний раз изменил сущность (системное поле)
    DELETE_TS timestamp,            --Когда удалено (системное поле)
    DELETED_BY varchar(50),          --Кем удалено (системное поле)
    primary key (ID)
)^
```

SQL-команды разделяются знаком "^". Это дает возможность задания сложных команд, содержащих внутри себя знаки ";".

Системные поля обязательны во всех сущностях.

Так как создаваемый справочник добавляется к уже существующей базе данных, необходимо воспользоваться автоматическим механизмом обновления БД. Для этого

создайте скрипт обновления в папке db\update\postgres\01\ модуля core вида "номер папки-порядок выполнения-название.sql". Также данный скрипт необходимо добавить в файл db\init\postgres\create-db.sql модуля core . Таким образом, в файле db\init\postgres\create-db.sql модуля core будет храниться последняя структура БД. А в db\update\postgres\ – скрипты обновления, чтобы привести обновляемую БД к последней версии.

Проверим, создается ли таблица в базе данных. Для этого выполните следующее:

1. Запустите командную строку в *рабочем каталоге* .
2. В командной строке введите команду

```
gradle restart
```

3. После выполнения команды откройте pgAdmin и подключитесь к базе данных **thesis\_имя\_расширения** . Убедитесь, что таблица **ext\_post** появилась в списке таблиц.

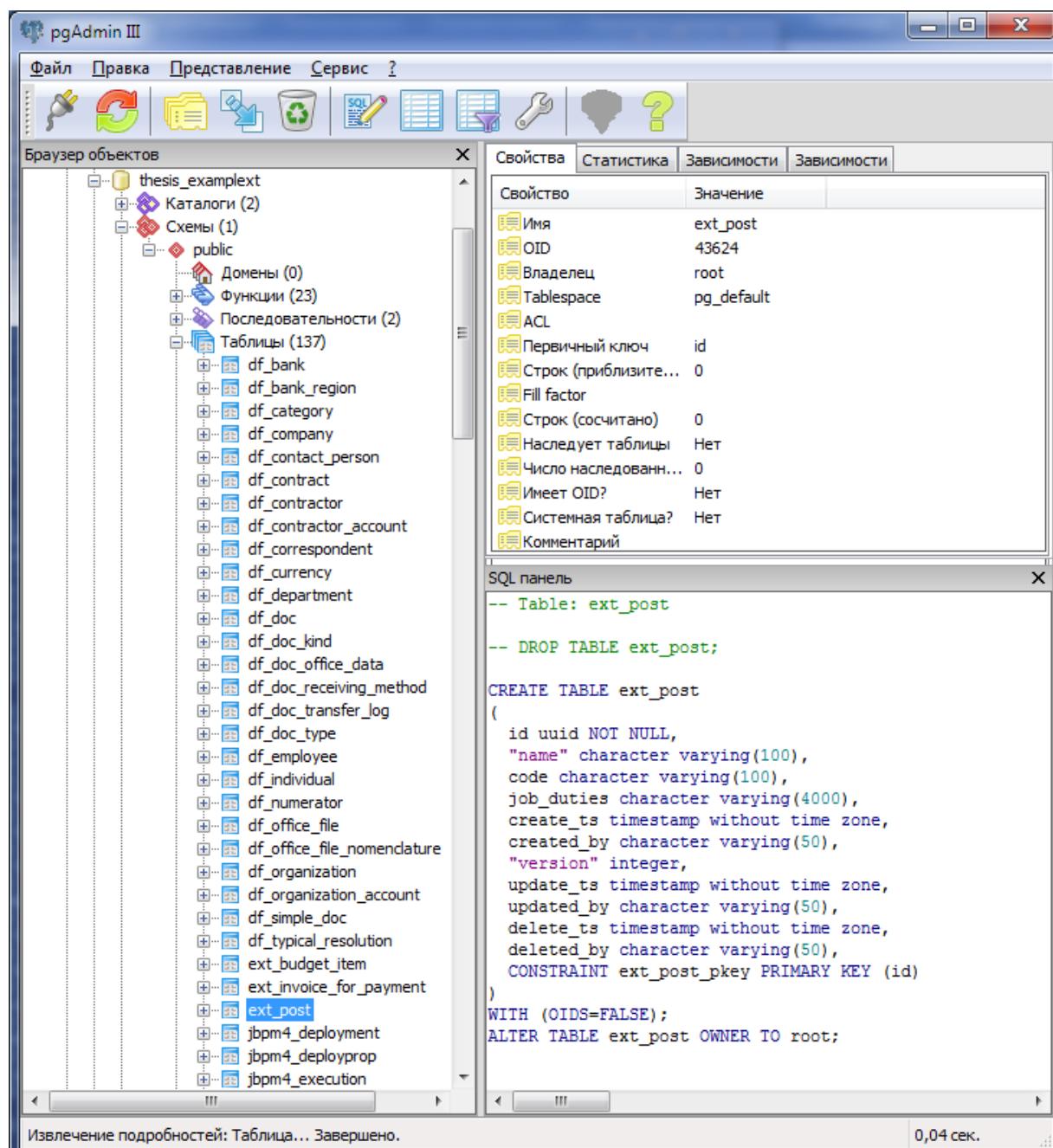


Рисунок 36. Таблицы приложения

### 3.2. Создание класса сущности

Предметная область моделируется в системе с помощью взаимосвязанных классов сущностей. Все персистентные (хранящиеся в БД) сущности должны находиться в пакете com.haulmont.ext.core.entity модуля global .

Классы сущности должны быть унаследованы от класса com.haulmont.cuba.core.entity.StandardEntity , который содержит в себе все системные поля. В результате класс Post будет выглядеть следующим образом:

```
package com.haulmont.ext.core.entity;

import com.haulmont.chile.core.annotations.NamePattern;
import com.haulmont.cuba.core.entity.StandardEntity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

// аннотируем класс как сущность предметной области
@Entity(name = "ext$Post")
// помечаем, что данный класс связан с таблицей в БД
@Table(name = "EXT_POST")
@NamePattern("%s|name")
public class Post extends StandardEntity {

    //помечаем атрибут name, что он связан с колонкой EXT_POST.NAME
    // и длина текстового поля <= 100
    @Column(name = "NAME", length = 100)
    private String name;

    @Column(name = "CODE", length = 100)
    private String code;

    @Column(name = "JOB_DUTIES", length = 4000)
    private String jobDuties;

    public String getName() {
        return name;
    }

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

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getJobDuties() {
        return jobDuties;
    }

    public void setJobDuties(String jobDuties) {
        this.jobDuties = jobDuties;
    }
}
```

### Совет

Для автоматической генерации методов доступа (get/set) можно воспользоваться сочетанием клавиш **Alt +Ins** , затем в меню выбрать **Getter and Setter** . В отобразившемся окне выберите требуемые поля и нажмите на кнопку **OK**.

### Внимание!

Методы доступа не должны содержать никакой логики кроме чтения/установки атрибутов данной или связанных с ней сущностей.

Классы сущностей должны быть соответствующим образом проаннотированы. Более подробно об аннотациях можно прочитать в документации по OpenJPA <http://openjpa.apache.org>

После создания класса зарегистрируйте его в файле `ext-persistence.xml` :

```
<class>com.haulmont.ext.core.entity.Post</class>
```

Для локализации названий свойств сущностей в том же пакете что и классы создайте файлы `messages.properties` и `messages_ru.properties` .

В этих файлах определяются строки с ключом `имя_сущности` для имени сущности и `имя_сущности.имя_атрибута` для имен атрибутов. Эти названия будут использованы при отображении списка экземпляров сущности и в окне редактирования сущности.

### Подсказка

Все исходные файлы, в том числе `*.properties` , должны иметь кодировку UTF-8 , поэтому в начале работы необходимо настроить кодировку UTF-8 в среде **IntelliJ IDEA** . Для этого зайдите в **File->Settings->Project Settings->File Encodings** . В выпадающем списке **Default encoding for properties files** выберите `UTF-8` . Если не выбрана опция **Autodetect UTF-encoded files** , то включите ее.

Например, для русской локализации сущности `Post` перевод будет следующим:

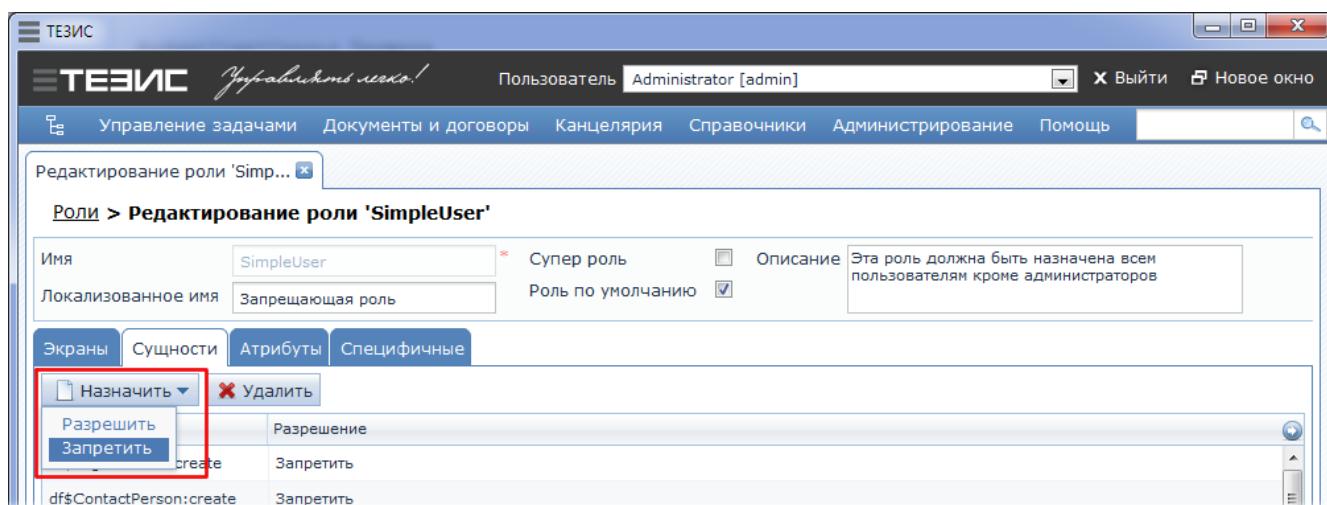
```

Post=Должность
Post.name=Наименование
Post.code=Код
Post.id=Идентификатор
Post.jobDuties=Должностные обязанности
  
```

Проверим, появилась ли созданная сущность в системе. Для этого пересоберите проект и зайдите в систему.

Возможность создавать новые должности должна быть не у каждого пользователя системы, а только у обладающего специальными правами. Для того чтобы запретить пользователям создавать или изменять определенные сущности системы, существует роль `SimpleUser`. Настроим роль таким образом, чтобы обладающий этой ролью пользователь не мог создавать сущности `Должность`.

Перейдите в режим редактирования роли `SimpleUser`, далее – на вкладку **Сущности**. Нажмите на кнопку **Назначить** и в выпадающем списке выберите пункт **Запретить**.



**Рисунок 37. Окно редактирования роли SimpleUser**

В отобразившемся окне в списке слева найдите и выделите сущность `ext$Post` и отметьте галочки `create`, `delete`, `update`.

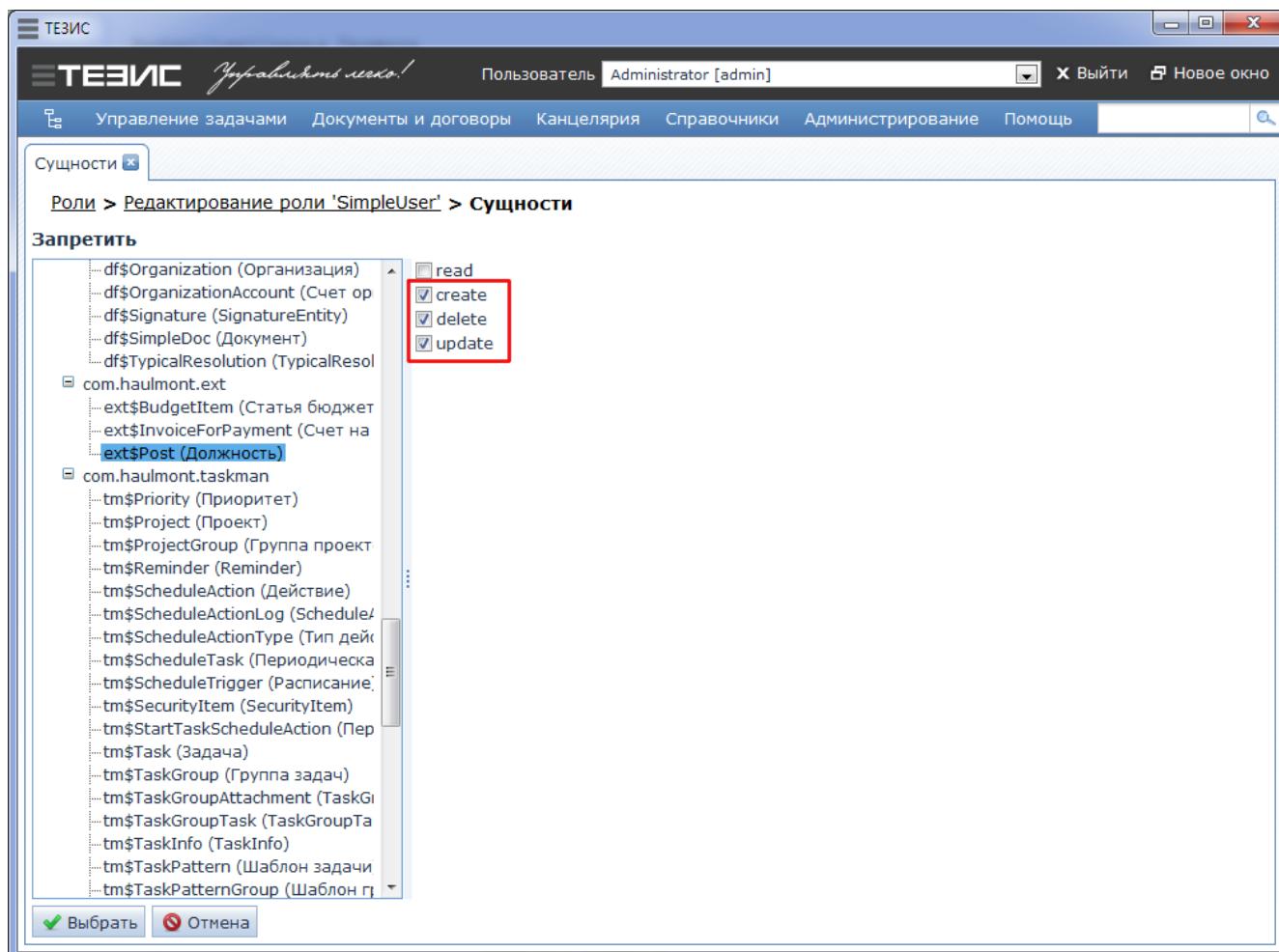


Рисунок 38. Окно редактирования роли SimpleUser

После этого нажмите на кнопку **Выбрать**.

### 3.3. Создание экрана со списком сущностей

В первую очередь создайте пакет com.haulmont.ext.web.ui.post в модуле web , где мы будем создавать файлы, относящиеся к экранам сущности Post .

Для создания списка "Должности" создайте следующие файлы:

1. XML-дескриптор списка сущностей post-browse.xml

Для того чтобы отобразить должности в таблице с возможностью создавать, изменять и удалять, необходимо чтобы файл post-browse.xml имел следующий вид:

```
<window
```

```

xmlns = "http://schemas.haulmont.com/cuba/5.2/window.xsd"
class = "com.haulmont.ext.web.ui.post.PostBrowser"
messagesPack = "com.haulmont.ext.web.ui.post"
caption = "msg://browserCaption"
lookupComponent = "postsTable"
>

<dsContext>
    <collectionDatasource id = "postsDs"
        class = "com.haulmont.ext.core.entity.Post"
        view = "_local">
        <query>
            select p from ext$Post p order by p.name
        </query>
    </collectionDatasource>
</dsContext>

<layout expandLayout = "true">
    <vbox id = "table-panel"
        expand = "postsTable"
        spacing = "true"
        height = "100%">
        <filter id = "genericFilter"
            datasource = "postsDs"
            stylename = "edit-area">
            <properties include = ".*"
                exclude = ""/>
        </filter>

        <table id = "postsTable"
            editable = "false">
            <buttonsPanel>
                <button action="postsTable.create" icon="icons/create.png"/>
                <button action="postsTable.edit" icon="icons/edit.png"/>
                <button action="postsTable.remove" icon="icons/remove.png"/>
                <button action="postsTable.excel" icon="icons/excel.png"/>
                <button action="postsTable.refresh" icon="icons/refresh.png"/>
            </buttonsPanel>
            <rowsCount/>
            <columns>
                <column id = "name"/>
                <column id = "code"/>
                <column id = "jobDuties"/>
            </columns>
            <rows datasource = "postsDs"/>
        </table>
    </vbox>
</layout>
</window>

```

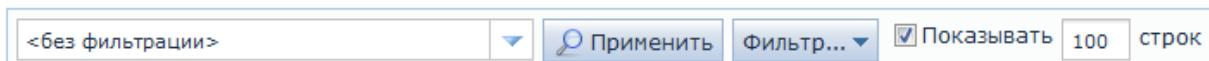
В элементе *dsContext* добавлен один компонент *источника данных collectionDatasource*, который выбирает сущности Post с помощью JPQL запроса `select p from ext$Post p order by p.name` (Подробнее о JPQL можно прочитать на сайте <http://openjpa.apache.org/>) с представлением `view= "_local"`.

Рассмотрим элементы дескриптора более подробно.

Компонент *Generic Filter* (XML-имя компонента – *filter*) служит для

отображения произвольного фильтра.

**Рисунок 39. Вид компонента Generic Filter**



Компонент *Table* (XML-имя компонента – *table* ) служит для отображения данных в виде сетки, упорядоченной в строки и столбцы.

1 строка		
Наименование	Код	Должностные обязанности
Инженер-программист (программист)	0001	Описание обязанностей

**Рисунок 40. Вид компонента Table**

Элемент *column* задает опции для колонки таблицы. Обязательно содержит атрибут *id* , в котором задается название атрибута сущности, выводимого в колонке. Название берется из *пакета локализованных сообщений* .

Для отображения количества строк таблицы используется элемент *rowCount*

Над таблицей находится панель, содержащая кнопки для управления данными в этой таблице. XML-имя такой панели – *buttonsPanel* .



**Рисунок 41. Вид компонента buttonsPanel**

Размещение перечисленных выше компонентов задается с помощью контейнера *BoxLayout* . Существует три типа этого контейнера, определяемых именем XML-элемента. В нашем дескрипторе используется *vbox* , в результате чего элементы расположены вертикально.

*Дескрипторы экранов* имеют идентификаторы, по которым удобно вызывать экраны из меню или из программного кода. Идентификаторы назначаются в файле *ext-web-screens.xml* модуля *web* .

```
<screen id="ext$Post.browse"
template="/com/haulmont/ext/web/ui/post/post-browse.xml"/>
```

Чтобы добавить экран в меню, измените файл ext-web-menu.xml модуля web следующим образом:

```
<menu-config>
    ...
    <menu id="ext">
        ...
        <item id="ext$Post.browse"/>
    </menu>
</menu-config>
```

Если Вы хотите добавить новый пункт меню в уже существующее меню, скопируйте содержимое файла thesis-web-menu.xml в файл ext-web-menu.xml и добавляйте созданный Вами пункт меню в существующую структуру. Например, чтобы добавить подпункт меню "**Должности**" в меню "**Справочники**", измените файл ext-web-menu.xml следующим образом:

```
<menu-config xmlns="http://www.haulmont.com/schema/cuba/gui/menu-config.xsd">
    ...
    <menu id="reference">
        <!-- Taskman -->
        ...
        <!-- Docflow -->
        ...

        <!-- Extension -->
        <item id="ext$Post.browse"/>
    </menu>
    ...
</menu-config>
```

Для того чтобы ссылки отображались в меню корректно, создайте файлы *главного пакета сообщений расширения* messages.properties и messages\_ru.properties в пакете com.haulmont.ext.web модуля web , прописав в них названия пункта меню на английском и русском языках соответственно.

```
menu-config.ext$Post.browse=Должности
```

Для локализации сообщений экрана создайте файлы messages.properties и messages\_ru.properties в пакете экрана com.haulmont.ext.web.ui.post . Например, для русской локализации файл будет иметь следующий вид:

```
browserCaption=Должности
```

2. Класс контроллера будет называться PostBrowser . Описание класса представлено ниже.

```
package com.haulmont.ext.web.ui.post;
```

```
import com.haulmont.cuba.gui.ComponentsHelper;
import com.haulmont.cuba.gui.components.AbstractLookup;
import com.haulmont.cuba.gui.components.IFrame;
import com.haulmont.cuba.gui.components.Table;
import com.haulmont.cuba.gui.components.actions.RefreshAction;
import com.haulmont.cuba.web.filestorage.WebExportDisplay;
import com.haulmont.cuba.web.gui.components.WebFilter;
import com.haulmont.thesis.web.actions.ThesisExcelAction;

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

public class PostBrowser extends AbstractLookup {

    @Inject
    protected Table postsTable;

    @Inject
    protected WebFilter genericFilter;

    public PostBrowser(IFrame frame) {
        super(frame);
    }

    @Override
    public void init(Map<String, Object> params) {
        super.init(params);
        ComponentsHelper.createActions(postsTable);
        postsTable.addAction(new RefreshAction(postsTable));
        postsTable.addAction(new ThesisExcelAction(postsTable,
            new WebExportDisplay(), genericFilter));
    }
}
```

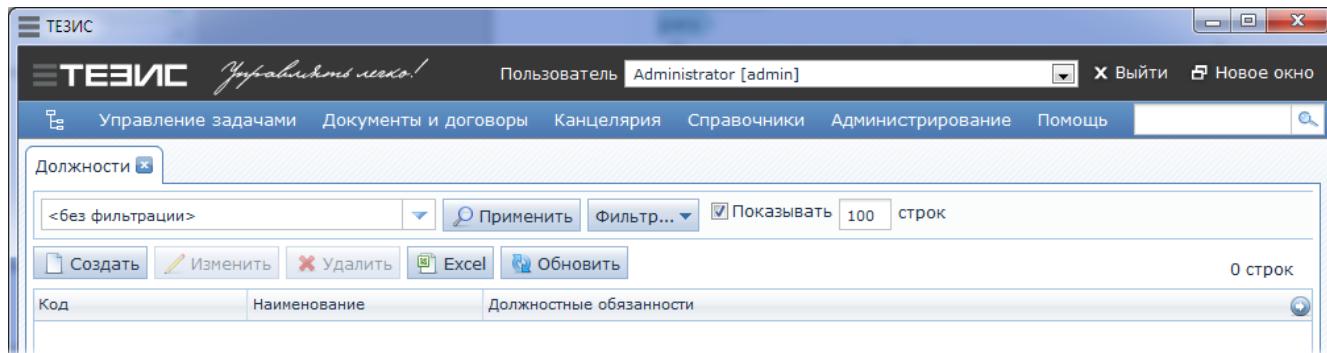
PostBrowser наследуется от AbstractLookup и переопределяет метод init(Map<String, Object> params), который вызывается при формировании окна. Переопределение метода необходимо для того, чтобы создать обработчики событий на кнопки, объявленные в post-browse.xml . Так как в большинстве списков необходимы стандартные операции (создание, изменение, удаление), воспользуемся классом ComponentsHelper со статическим методом createActions в который передается компонент таблицы, к которому привязаны действия.

Обратите внимание, в классе контроллера был использован механизм **внедрения зависимостей** (англ. Dependency injection ) для получения ссылок на используемые объекты.

```
@Inject
protected Table postsTable;
```

Согласно этому механизму в текущем экране будет произведен поиск таблицы с соответствующим именем.

Посмотрим, как созданный нами экран выглядит в системе. Для этого пересоберите проект и зайдите в систему. Откройте пункт меню **Справочники -> Должности**.



**Рисунок 42. Окно управления должностями**

Можно убедиться, что данный экран содержит следующие элементы:

- Панель с фильтром.
- Панель с кнопками **Создать**, **Изменить**, **Удалить**, **Обновить**, **Excel** и с меткой, отображающей число записей в таблице.
- Таблицу с колонками "**Код**", "**Наименование**", "**Должностные обязанности**"

### 3.4. Создание экрана редактирования сущности

Экран редактирования сущности "Должность" состоит из:

1. **XML-дескриптора** `post-edit.xml`. Для того чтобы экран редактирования сущности "Должность" имел поля для ввода наименования, кода и должностных обязанностей, XML-дескриптор должен иметь следующий вид:

```

<window
    xmlns = "http://schemas.haulmont.com/cuba/5.2/window.xsd"
    class = "com.haulmont.ext.web.ui.post.PostEditor"
    caption = "msg://editorCaption"
    datasource = "postDs"
    messagesPack = "com.haulmont.ext.web.ui.post"
    focusComponent = "fieldGroup">

    <dsContext>
        <datasource id = "postDs"
            class = "com.haulmont.ext.core.entity.Post"
            view = "_local"/>
    </dsContext>

    <layout expand = "windowActions">
        <vbox spacing = "true">
            <fieldGroup id = "fieldGroup">

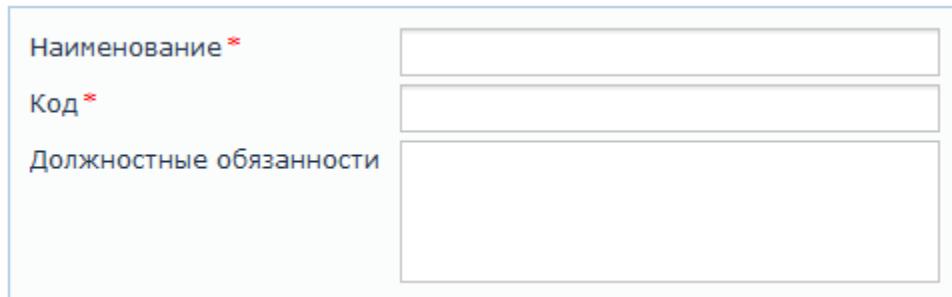
```

```

    datasource = "postDs"
    width="100%"
    stylename="edit-area">
<column width = "270px">
    <field id = "name"
           required = "true"/>
    <field id = "code"
           required = "true"/>
    <field id="jobDuties"
           rows="5"/>
</column>
</fieldGroup>
</vbox>
<iframe id = "windowActions"
         src = "/com/haulmont/cuba/gui/edit-window.actions.xml"/>
</layout>
</window>
  
```

В экран редактирования будет передаваться одна сущность либо на создание, либо на изменение и проставляться в *datasource*. Поэтому вместо *collectionDatasource* объявлен *datasource*.

Для отображения набора атрибутов редактируемой или создаваемой сущности используется компонент **FieldGroup** (XML-имя компонента – *fieldGroup* ).



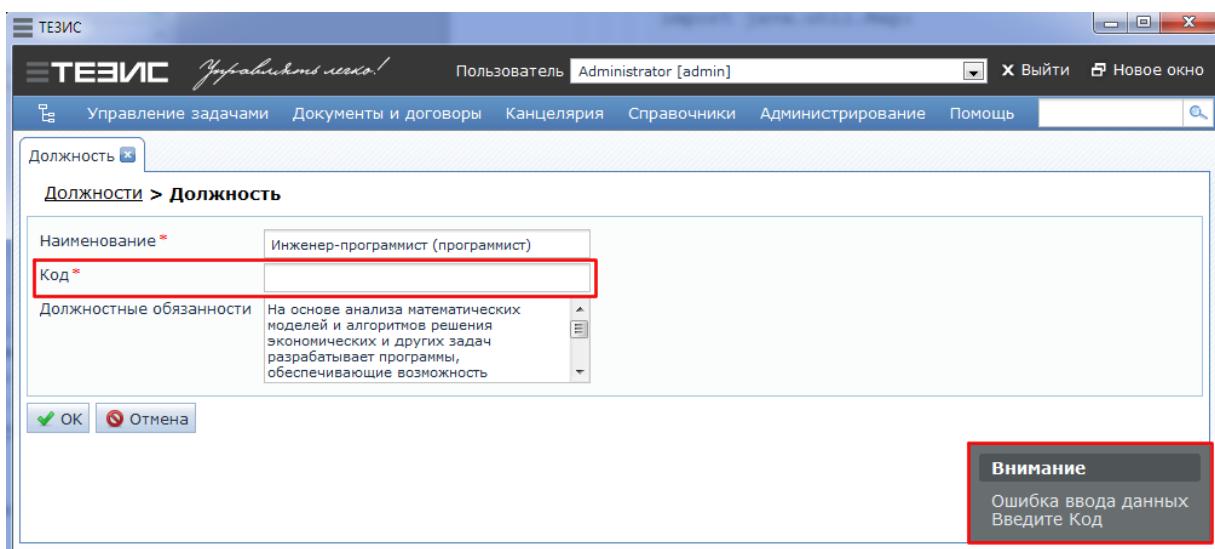
**Рисунок 43. Вид компонента FieldGroup**

Данный компонент содержит в себе два элемента *field* :

```

<field id = "name"
       required = "true"/>
<field id = "code"
       required = "true"/>
<field id="jobDuties"
       rows="5"/>
  
```

Обратите внимание, что в первых двух элементах *field* указан атрибут *required = "true"*. Данный атрибут указывает, что в данное поле обязательно должно быть введено значение. Такое поле помечается красной звездочкой. В случае если значение в данное поле не было введено, пользователю выводится сообщение.



**Рисунок 44.**

*XML-дескриптор* экрана редактирования должен содержать фрейм с кнопками подтверждения или отказа операции над сущностью. Есть два стандартных варианта такого фрейма:

1. /com/haulmont/cuba/gui/edit-window.actions.xml – фрейм с двумя кнопками: **OK**, **Cancel**. При нажатии на кнопку **OK** происходит фиксация операции, и окно закрывается.
2. /com/haulmont/cuba/gui/extended-edit-window.actions.xml – фрейм с тремя кнопками: **OK**, **OK&Close**, **Cancel**. В этом фрейме есть возможность подтвердить операцию, не закрывая окна.

Зарегистрируйте экран в файле `ext-web-screens.xml` модуля `web`, чтобы его можно было открывать из меню, других окон или из программного кода.

```
<screen id="ext$Post.edit"
template="/com/haulmont/ext/web/ui/post/post-edit.xml"/>
```

Для корректного отображения название редактируемой сущности нужно добавить **локализованное название** сущности в файлы `messages.properties` и `messages_ru.properties` в пакете `com.haulmont.ext.web.ui.post`. Например, для русской локализации в файл будет добавлена следующая строка:

```
editorCaption=Должность
```

2. Класс контроллера для редактирования сущности должен быть унаследован

от `com.haulmont.cuba.gui.components.AbstractEditor`, который реализует интерфейс `Window.Editor`.

Класс контроллера будет иметь следующий вид:

```
package com.haulmont.ext.web.ui.post;

import com.haulmont.cuba.core.entity.Entity;
import com.haulmont.cuba.gui.components.AbstractEditor;
import com.haulmont.cuba.gui.components.IFrame;

import java.util.Map;

public class PostEditor extends AbstractEditor {

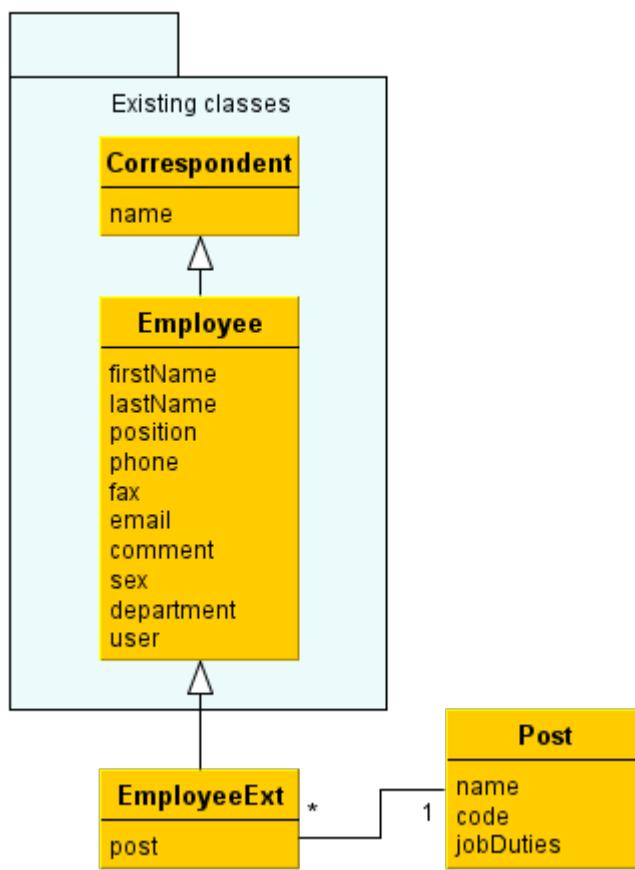
    public PostEditor(IFrame frame) {
        super(frame);
    }

    @Override
    public void setItem(Entity item) {
        super.setItem(item);
    }

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

### 3.5. Расширение существующей функциональности

В системе **ТЕЗИС** существует справочник сотрудников. Сущность `Сотрудник` имеет поле **Должность** текстового типа. Требуется изменить поле, сделав его ссылочным на сущность `Должность`, которая уже создана в системе (см. пункт 3.1. данного Руководства). Очевидно, необходимо создать новую сущность `EmployeeExt`, которая будет унаследована от сущности `Employee`, и добавить в нее атрибут `Должность`. При этом потребуется создавать и новую таблицу в базе данных. Это связано с тем, что таблица `Сотрудник` (`Employee`) не является корневой в иерархии наследования. Корневой таблицей является таблица `Корреспондент` (`Correspondent`). В корневой сущности используется стратегия наследования `InheritanceType.JOINED`. Такая стратегия наследования использует разные таблицы для каждого класса иерархии. В итоге мы должны получить следующую структуру классов:



**Рисунок 45. Диаграмма структуры классов**

Опишем более подробно порядок действий.

### Расширение сущности

Создайте в проекте сущность, унаследованную от сущности Employee и добавьте в нее атрибут post :

```

@Entity(name = "ext$Employee")
@Table(name = "EXT_EMPLOYEE")
@PrimaryKeyJoinColumn(name="EMPLOYEE_ID", referencedColumnName = "CORRESPONDENT_ID")
@DiscriminatorValue("A")
@NamePattern("%s|name")
public class EmployeeExt extends Employee{

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "POST_ID")
    protected Post post;

    public Post getPost() {
        return post;
    }

    public void setPost(Post post) {
        this.post = post;
    }
}
  
```

```
}
```

Новая сущность должна иметь новое имя в аннотации `@Entity`. Также должна быть перенесена из базовой сущности аннотация `@NamePattern` (если она определена). Аннотация `@Table` будет определяться базовой сущностью (при условии что новая таблица не будет создана). Аннотацию `@Listeners`, если она указана для базовой сущности, дублировать не нужно. Так как мы будем создавать новую таблицу, нужно указать аннотации `@Table`, `@PrimaryKeyJoinColumn` и `@DiscriminatorValue`.

После создания класса зарегистрируйте его в файле `ext-persistence.xml`:

```
<class>com.haulmont.ext.core.entity.EmployeeExt</class>
```

Для того чтобы сделать возможным создание наследуемой сущности вместо базовой, необходимо зарегистрировать замещение сущности `Employee` сущностью `EmployeeExt` в файле `ext-metadata.xml` модуля `global` в элементе `entityFactory`:

```
&lt;metadata&gt;
    &lt;entityFactory&gt;
        &lt;replace class="com.haulmont.thesis.core.entity.Employee"
                   with="com.haulmont.ext.core.entity.EmployeeExt"/&gt;
    &lt;/entityFactory&gt;
&lt;/metadata&gt;
```

Далее создайте *представления* для новой сущности в файле `ext-views.xml` модуля `core`:

```
<view class="com.haulmont.ext.core.entity.EmployeeExt"
      name="browse"
      extends="_local">
    <property name="post"
              view="_minimal"/>
</view>

<view class="com.haulmont.ext.core.entity.EmployeeExt"
      name="edit"
      extends="_local">
    <property name="post"
              view="_minimal"/>
</view>
```

## Модификация базы данных

Создайте новый скрипт для создания таблицы `EXT_EMPLOYEE`.

```
create table EXT_EMPLOYEE (
    EMPLOYEE_ID uuid,
    POST_ID uuid,
    primary key (EMPLOYEE_ID)
)^
alter table EXT_EMPLOYEE add constraint FK_EXT_EMPLOYEE_DF_EMPLOYEE
    foreign key (EMPLOYEE_ID) references DF_EMPLOYEE(CORRESPONDENT_ID)
```

```
^
alter table EXT_EMPLOYEE add constraint FK_EXT_EMPLOYEE_EXT_POST
    foreign key (POST_ID) references EXT_POST(ID)
^
```

Добавьте данный скрипт в файл `create-db.sql` модуля `core`.

## Модификация экранов

Дескрипторы экранов поддерживают наследование путем указания в корневом элементе атрибута `extends`, содержащего ссылку на базовый дескриптор.

Правила переопределения элементов экрана:

- если в расширяющем дескрипторе указан некоторый элемент, в базовом дескрипторе будет произведен поиск соответствующего элемента по следующему алгоритму:
  - если переопределяющий элемент – `view`, то ищется соответствующий элемент по атрибутам `name`, `class`, `entity`;
  - если переопределяющий элемент – `property`, то ищется соответствующий элемент по атрибуту `name`;
  - в других случаях если в переопределяющем элементе указан атрибут `id`, то ищется соответствующий элемент с таким же `id`;
  - если поиск дал результат, то найденный элемент переопределяется;
  - если поиск не дал результата, то в базовом дескрипторе подсчитывается количество элементов с данным именем по данному пути. Если число таких элементов не превышает одного, то найденный элемент переопределяется;
  - если поиск не дал результата, то в базовом дескрипторе подсчитывается количество элементов с данным именем по данному пути. Если число таких элементов равно нулю или их больше одного, то происходит добавление нового элемента;
- в переопределяемом либо добавляемом элементе устанавливается текст из расширяющего элемента;
- в переопределяемый либо добавляемый элемент копируются все атрибуты из расширяющего элемента.

Добавление нового элемента по умолчанию производится в конец списка соседних

элементов. Если необходимо добавить новый элемент в начало или с произвольным индексом, нужно сделать следующие действия:

- определить в расширяющем дескрипторе *namespace* :

```
  xmlns:ext="http://www.haulmont.com/schema/cuba/gui/window-ext.xsd"
```

- добавить в переопределяющий элемент атрибут *ext:index* с желаемым индексом, например: *ext:index="0"* .

Расширение экрана браузера сотрудников для отображения новой колонки выглядит следующим образом:

```
<window
    xmlns:ext="http://www.haulmont.com/schema/cuba/gui/window-ext.xsd"
    extends="/com/haulmont/thesis/web/ui/employee/employee-browse.xml"
>

<dsContext>
    <collectionDatasource
        id="employeesDs"
        class="com.haulmont.ext.core.entity.EmployeeExt"
        view="browse"
    >
        <query><![CDATA[
            select e from ext$Employee e
            <#if param$departmentLookup?has_content>
                where e.department is null</#if> order by e.name
            ]]>
        </query>
    </collectionDatasource>
</dsContext>
<layout expandLayout="true">
    <vbox id="table-panel" expand="employeesTable" spacing="true">
        <groupTable id="employeesTable">
            <columns>
                <column id="position"
                    visible ="false"/>
                <column id="post"
                    ext:index="5"/>
            </columns>
        </groupTable>
    </vbox>
</layout>
</window>
```

Обратите внимание, мы переопределили колонку **position** , сделав ее невидимой: *visible = "false"* , – и добавили новую колонку **post** . Также был переопределен источник данных, теперь в нем выгружаются экземпляры сущности *EmployeeExt* .

Расширение экрана редактора сотрудника для отображения нового поля выглядит следующим образом:

```
<window
    xmlns:ext="http://www.haulmont.com/schema/cuba/gui/window-ext.xsd"
    extends="/com/haulmont/thesis/web/ui/employee/employee-edit.xml"
>
```

```

<dsContext>
    <datasource id="employeeDs"
        class="com.haulmont.ext.core.entity.EmployeeExt"
        view="edit"/>
    <collectionDatasource id="postDs"
        class="com.haulmont.ext.core.entity.Post"
        view="_local"/>
</dsContext>
<#assign width=300>
<layout>
    <tabsheet id="tabsheet">
        <tab id="mainTab">
            <scrollbox>
                <vbox>
                    <grid id="grid">
                        <rows>
                            <row id="positionRow"
                                visible="false"/>
                            <row id="postRow"
                                ext:index="5">
                                <label value="msg://position"/>
                                <pickerField id="post"
                                    datasource="employeeDs"
                                    property="post"
                                    lookupScreen="ext$Post.browse"
                                    width="${width}"/>
                            </row>
                        </rows>
                    </grid>
                </vbox>
            </scrollbox>
        </tab>
    </tabsheet>
</layout>
</window>

```

Обратите внимание, был переопределен источник данных `employeeDs` , а также добавлен новый источник данных для отображения экземпляров сущности Должность .

## Регистрация экранов

Новые экраны необходимо зарегистрировать в `ext-web-screens.xml` модуля `web` , переопределив также алиас базовых экранов. Это даст возможность вызывать новые экраны не зная о них:

```

<screen id="df$Employee.browse"
    template="/com/haulmont/ext/web/ui/extempLOYEE/extempLOYEE-browse.xml"/>
<screen id="df$Employee.edit"
    template="/com/haulmont/ext/web/ui/extempLOYEE/extempLOYEE-edit.xml"/>
<screen id="ext$Employee.edit"
    template="/com/haulmont/ext/web/ui/extempLOYEE/extempLOYEE-edit.xml"/>

```

Посмотрим, как изменился экран редактирования сотрудника:

Пользователь системы	<input type="text"/>
Фамилия	<input type="text"/> *
Имя	<input type="text"/>
Отчество	<input type="text"/>
Отображаемое имя	<input type="text"/> *
Подразделение	<input type="text"/>
Должность	<input type="text"/> <span style="border: 2px solid red; padding: 2px;"> </span> <span style="color: green; font-size: 2em; vertical-align: middle;">→</span> <span style="border: 2px solid red; padding: 2px;"> </span> <span style="color: blue; font-size: 1.5em; vertical-align: middle;">✖</span>
E-mail	<input type="text"/>
Телефон	<input type="text"/>
Факс	<input type="text"/>
Комментарий	<input type="text"/>
Пол	<input checked="" type="radio"/> Мужской <input type="radio"/> Женский

Пользователь системы	<input type="text"/>
Фамилия	<input type="text"/> *
Имя	<input type="text"/>
Отчество	<input type="text"/>
Отображаемое имя	<input type="text"/> *
Подразделение	<input type="text"/>
Должность	<input type="text"/> <span style="border: 2px solid red; padding: 2px;"> </span> <span style="color: green; font-size: 2em; vertical-align: middle;">→</span> <span style="border: 2px solid red; padding: 2px;"> </span> <span style="color: blue; font-size: 1.5em; vertical-align: middle;">✖</span>
E-mail	<input type="text"/>
Телефон	<input type="text"/>
Факс	<input type="text"/>
Комментарий	<input type="text"/>
Пол	<input checked="" type="radio"/> Мужской <input type="radio"/> Женский

**Рисунок 46. Окно редактирования сотрудника**

## Глава 4. Разработка собственного проекта расширения

Предположим, в системе **ТЕЗИС** потребовалась возможность создавать новый документ – счет на оплату . Счета на оплату должны отображаться в системе как список. Необходимо предусмотреть возможность удаления и редактирования документа "Счет на оплату", а также возможность быстрого отбора по статусу документа. Кроме того что документ поступает в систему, он должен пройти несколько этапов, перед тем, как он будет оплачен. *Инициатор* создает счет на оплату в системе и запускает процесс оплаты счета. *Руководитель отдела* может утвердить счет на оплату, отменить процесс оплаты данного документа или отправить его на доработку тому лицу, которое приспало данный документ. После того как руководитель отдела утвердит данный документ, счет на оплату должен пройти согласование финансовой службы. *Финансист* может отменить процесс оплаты данного документа, согласовать документ или отправить его обратно руководителю отдела, если возникли какие-либо вопросы. После согласования финансистом документ должен поступить в бухгалтерию на проверку главным бухгалтером. *Главный бухгалтер* может отправить документ в оплату, а может вернуть руководителю отдела. После отправки документа в оплату он поступает бухгалтеру. *Бухгалтер* может оплатить документ или не оплачивать, а вернуть главному бухгалтеру на доработку. После оплаты документа инициатору должно прийти сообщение об успешной оплате счета.

Переведем данную задачу в терминологию системы.

### Постановка задачи разработчику

В проекте приложения создать новую карточку "Счет на оплату". Карточка должна иметь следующие вкладки:

- Детали
- Проекты
- Вложения
- Процессы
- Обсуждения
- Иерархия

- История
- Безопасность

Вкладка **Детали** должна содержать следующие атрибуты:

Имя атрибута	Тип атрибута
Статья бюджета	Справочник. Статьи бюджета
Контрагент	Справочник. Контрагенты
Основание	Список существующих задач, документов, договоров, счетов на оплату
Дата оплаты	Дата
Номер счета	Строка (создать нумератор вида "СО-00001")
Сумма	Число
Описание платежа	Строка

Создать экран, отображающий список счетов на оплату и содержащий следующие элементы:

- Таблицу, содержащую карточки типа "Счет на оплату". Таблица должна иметь колонки **Вложения**, **Сейчас у**, **Статья бюджета**, **Контрагент**, **Основание**, **Дата оплаты**, **Сумма**, **Валюта**, **Номер**, **Описание платежа**, **Тек. процесс**, **Тек. состояние** и иметь возможность динамической группировки по любому полю.
- Фильтр для поиска по заданным условиям.
- Кнопку **Создать**, позволяющую создать счет на оплату двумя способами, открывающимися в выпадающем списке, а именно: создать новый счет на оплату с помощью кнопки **Новый** и создать на основе копирования существующего счета на оплату с помощью кнопки **Копировать**.
- Кнопки **Изменить**, **Удалить**, **Обновить**, **Excel**.
- Флажок **Скрыть журнал действий**, определяющий, отображается

ли для выбранной задачи журнал действий (в нижней части экрана).

- Текстовое поле, содержащее информацию о количестве карточек в таблице.

Создать роль, необходимую для создания карточек счета на оплату, задать ограничения для существующих групп доступа. Создать *процесс* "Оплата счета" и новые роли для этого процесса. Создать новые *папки приложения* для процесса "Оплата счета".

Для реализации справочника статей бюджета создать новую сущность Статья бюджета с полями **Наименование** и **Код**. Создать экран, отображающий список статей бюджета, с кнопками **Создать**, **Удалить**, **Изменить**, **Обновить**, **Excel**. Создать экран редактирования статьи бюджета.

## 4.1. Создание нового справочника

В первую очередь создайте справочник "Статьи бюджета".

Создание любой сущности в системе состоит из следующих этапов:

1. Создание таблицы в БД.
2. Создание класса Entity, отвечающего за объектное представление таблицы БД в системе.
3. Создание списка сущностей.
4. Создание окна редактирования сущности.

### 4.1.1. Создание таблицы "Статьи бюджета" в БД

Таблицы, относящиеся к разрабатываемому расширению, должны начинаться с префикса EXT. В данном случае этот справочник будет относиться к счетам на оплату и будет иметь два поля: "**Наименование**" и "**Код**". Скрипт создания таблицы будет выглядеть следующим образом:

```
create table EXT_BUDGET_ITEM (
    ID uuid,                      --Первичный ключ (системное поле)
    NAME varchar(100),              --Наименование
    CODE varchar(100),              --Код
    CREATE_TS timestamp,            --Когда создано (системное поле)
```

```
CREATED_BY varchar(50), --Кем создано (системное поле)
VERSION integer, --Версия (системное поле)
UPDATE_TS timestamp, --Когда было последнее изменение (системное поле)
UPDATED_BY varchar(50), --Кто последний раз изменил сущность (системное поле)
DELETE_TS timestamp, --Когда удалено (системное поле)
DELETED_BY varchar(50), --Кем удалено (системное поле)
primary key (ID)
)^
```

SQL-команды разделяются знаком "^". Это дает возможность задания сложных команд, содержащих внутри себя знаки ";".

Системные поля обязательны во всех сущностях.

Добавьте этот скрипт в файл db\init\postgres\create-db.sql модуля core .

### Совет

Чтобы легко найти файл в проекте, нажмите сочетание клавиш **Ctrl + Shift +N** и напишите имя файла.

Если необходимо изменить БД, которая уже находится в промышленной эксплуатации, то необходимо воспользоваться автоматическим механизмом обновления БД. Для этого создайте скрипт обновления в папке db\update\postgres\01\ модуля core вида "номер папки-порядок выполнения-название.sql" (Например, в нашем случае 01-001-addBudgetItem.sql). Таким образом, в файле db\init\postgres\create-db.sql модуля core будет храниться последняя структура БД. А в db\update\postgres\ – скрипты обновления, чтобы привести обновляемую БД к последней версии.

Проверим, создается ли таблица в базе данных. Для этого выполните следующее:

1. Запустите командную строку в *рабочем каталоге* .
2. В командной строке введите команду

```
gradle restart
```

3. После выполнения команды откройте pgAdmin и подключитесь к базе данных **thesis\_имя\_расширения** . Убедитесь, что таблица ext\_post появилась в списке таблиц.

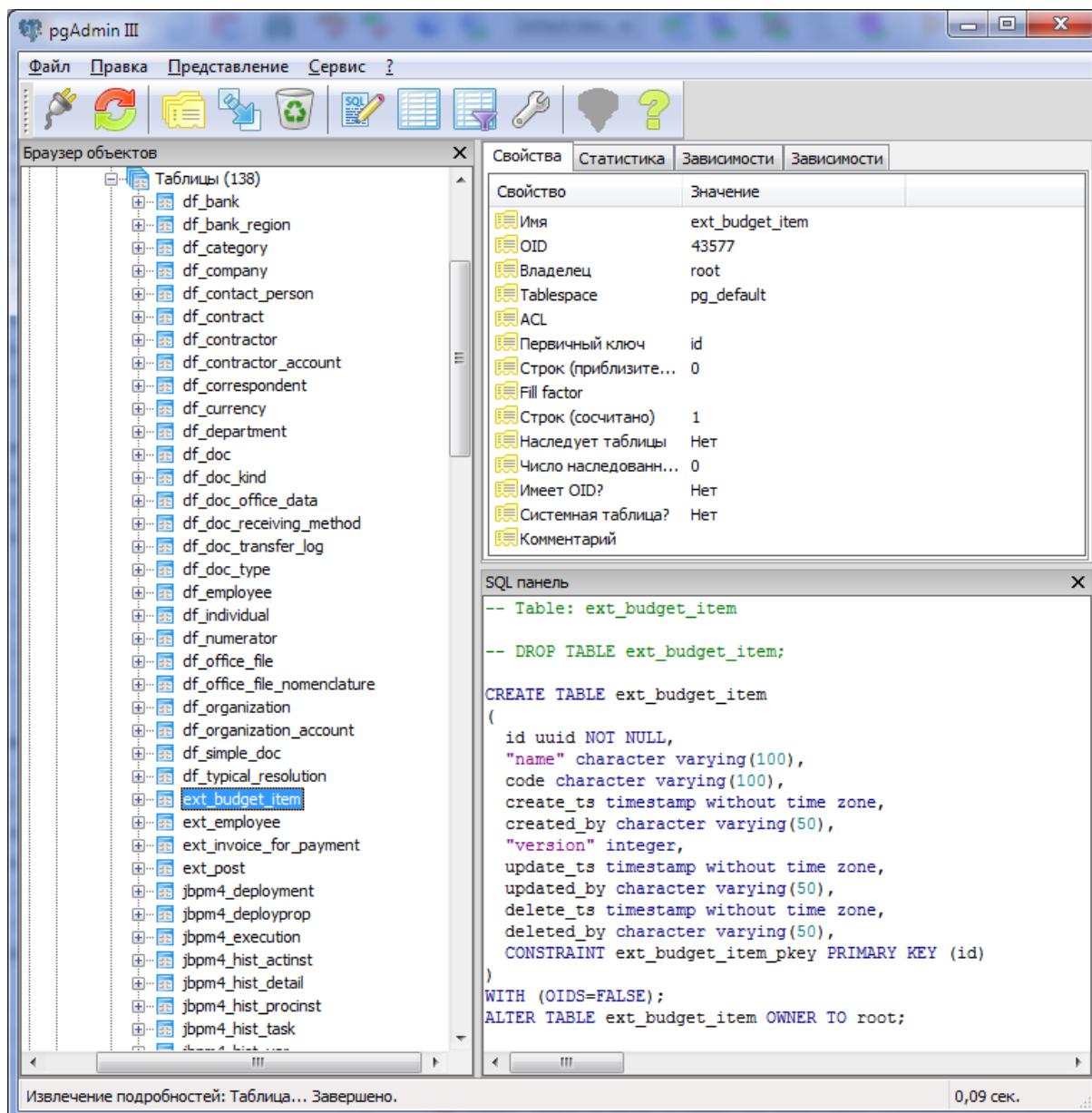


Рисунок 47. Таблицы базы данных

#### 4.1.2. Создание класса сущности BudgetItem

Предметная область моделируется в системе с помощью взаимосвязанных классов сущностей. Все персистентные (хранящиеся в БД) сущности должны находиться в пакете com.haulmont.ext.core.entity модуля global .

Классы сущности должны быть унаследованы от класса com.haulmont.cuba.core.entity.StandardEntity , который содержит в себе все

системные поля. В результате класс `BudgetItem` будет выглядеть следующим образом:

```
package com.haulmont.ext.core.entity;

import com.haulmont.chile.core.annotations.NamePattern;
import com.haulmont.cuba.core.entity.StandardEntity;
import javax.persistence.*;

import javax.persistence.Entity;

// аннотируем класс как сущность предметной области
@Entity(name = "ext$BudgetItem")

// помечаем, что данный класс связан с таблицей в БД
@Table(name = "EXT_BUDGET_ITEM")
@NamePattern("%s|name")
public class BudgetItem extends StandardEntity {

    private static final long serialVersionUID = 5672621802565606640L;

    //помечаем атрибут name, что он связан с колонкой EXT_BUDGET_ITEM.NAME
    // и длина текстового поля <= 100
    @Column(name = "NAME", length = 100)
    private String name;

    @Column(name = "CODE", length = 100)
    private String code;

    public String getName() {
        return name;
    }

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

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }
}
```

### Совет

Для автоматической генерации методов доступа (`get/set`) можно воспользоваться сочетанием клавиш **Alt +Ins** , затем в меню выбрать **Getter and Setter** . В отобразившемся окне выберите требуемые поля и нажмите на кнопку **OK**.

**Внимание!**

Методы доступа не должны содержать никакой логики кроме чтения/установки атрибутов данной или связанных с ней сущностей.

Классы сущностей должны быть соответствующим образом проаннотированы. Более подробно об аннотациях можно прочитать в документации по OpenJPA <http://openjpa.apache.org>

После создания класса зарегистрируйте его в файле `ext-persistence.xml`:

```
<class>com.haulmont.ext.core.entity.BudgetItem</class>
```

Для локализации названий свойств сущностей в том же пакете что и классы создайте файлы `messages.properties` и `messages_ru.properties`.

В этих файлах определяются строки с ключом `имя_сущности` для имени сущности и `имя_сущности.имя_атрибута` для имен атрибутов. Эти названия будут использованы при отображении списка экземпляров сущности и в окне редактирования сущности.

**Подсказка**

Все исходные файлы, в том числе `*.properties`, должны иметь кодировку UTF-8, поэтому в начале работы необходимо настроить кодировку UTF-8 в среде IntelliJ IDEA. Для этого зайдите в **File->Settings->Project Settings->File Encodings**. В выпадающем списке **Default encoding for properties files** выберите UTF-8. Если не выбрана опция **Autodetect UTF-encoded files**, то включите ее.

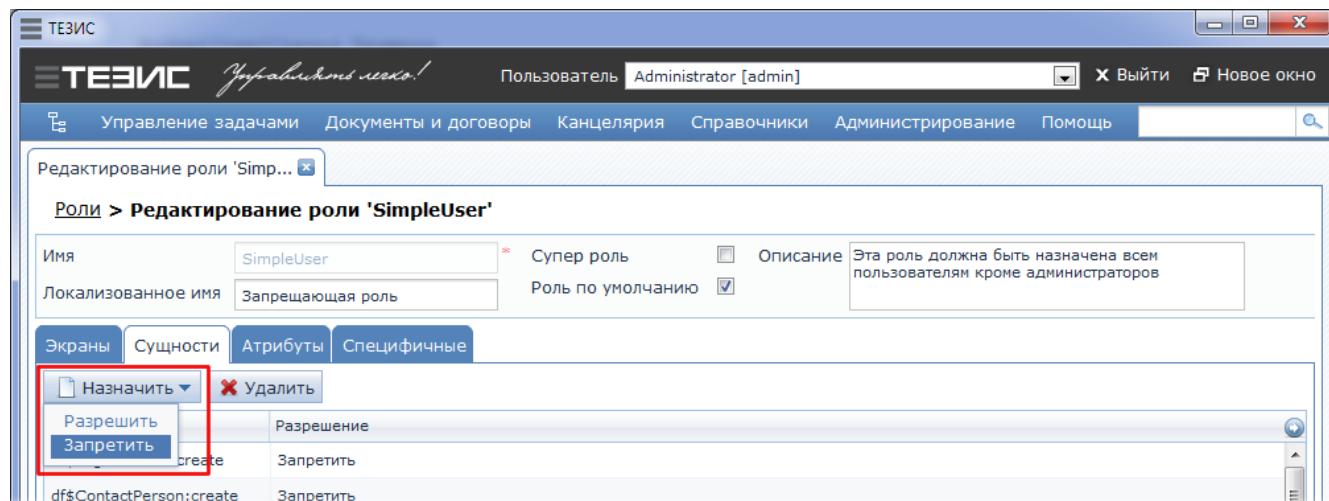
Например, для русской локализации `BudgetItem` перевод будет следующим:

```
BudgetItem=Статья бюджета
BudgetItem.name=Наименование
BudgetItem.code=Код
```

Возможность создавать новые статью бюджета должна быть не у каждого пользователя системы, а только у обладающего специальными правами. Для того чтобы запретить пользователям создавать или изменять определенные сущности

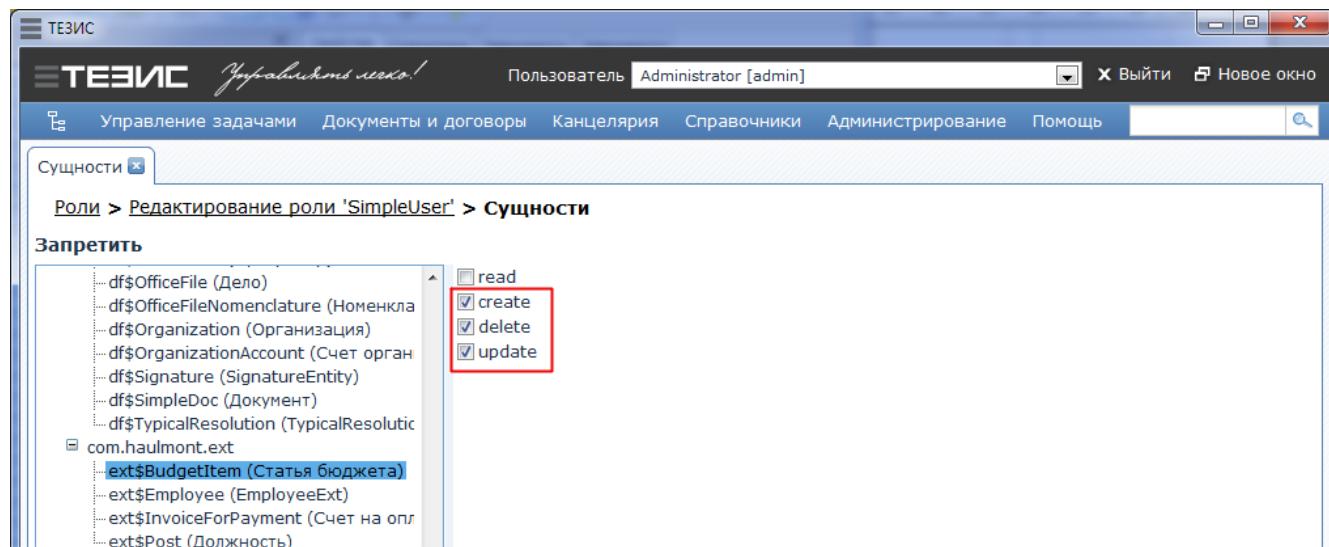
системы, существует роль *SimpleUser*. Настроим роль таким образом, чтобы обладающий этой ролью пользователь не мог создавать сущности Статья бюджета .

Перейдите в режим редактирования роли *SimpleUser* , далее – на вкладку **Сущности** . Нажмите на кнопку **Назначить** и в выпадающем списке выберите пункт **Запретить** .



**Рисунок 48. Окно редактирования роли SimpeUser**

В отобразившемся окне в списке слева найдите и выделите сущность *ext\$BudgetItem* и отметьте галочки **create** , **delete** , **update** .



**Рисунок 49. Окно редактирования роли SimpeUser**

После этого нажмите на кнопку **Выбрать**.

### 4.1.3. Создание экранов

Файлы, относящиеся к экранам, необходимо создавать в пакете com.haulmont.ext.web.ui модуля web .

Любой экран состоит из:

1. *XML-дескриптора* , описывающего источники данных и расположение визуальных компонентов экрана.
2. *Контроллера экрана* – Java или Groovy класса, в котором можно реализовывать бизнес-логику и управлять поведением компонентов, описанных в xml-дескрипторе, с помощью различных событий.

#### Создание списка сущностей

В первую очередь создайте пакет com.haulmont.ext.web.ui.budgetitem в модуле web , где мы будем создавать файлы, относящиеся к экранам сущности BudgetItem .

Для создания списка "Статьи бюджета" создайте следующие файлы:

1. *XML-дескриптор* списка сущностей budgetitem-browse.xml

Для того чтобы отобразить коды статей по бюджету в таблице с возможностью создавать, изменять и удалять, необходимо чтобы файл budgetitem-browse.xml имел следующий вид:

```
<window
    xmlns="http://schemas.haulmont.com/cuba/5.2/window.xsd"
    class="com.haulmont.ext.web.ui.budgetitem.BudgetItemBrowser"
    messagesPack="com.haulmont.ext.web.ui.budgetitem"
    caption="msg://browserCaption"
    lookupComponent="budgetItemsTable"
>

<dsContext>
    <collectionDatasource
        id="budgetItemsDs"
        class="com.haulmont.ext.core.entity.BudgetItem"
        view="_local">
        <query>
            select d from ext$BudgetItem d order by d.name
        </query>
    </collectionDatasource>
</dsContext>

<layout expandLayout="true">
    <vbox id="table-panel"
        expand="budgetItemsTable"
        spacing="true">
```

```

        height="100%">
<filter id="genericFilter"
        datasource="budgetItemsDs"
        stylename="edit-area">
    <properties include=".*"
                exclude="" />
</filter>

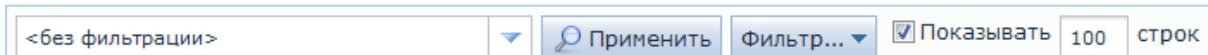
<table id="budgetItemsTable"
        editable="false">
    <buttonsPanel id ="buttonsPanel">
        <button action="budgetItemsTable.create"
                icon="icons/create.png"/>
        <button action="budgetItemsTable.edit"
                icon="icons/edit.png"/>
        <button action="budgetItemsTable.remove"
                icon="icons/remove.png"/>
        <button action="budgetItemsTable.excel"
                icon="icons/excel.png"/>
        <button action="budgetItemsTable.refresh"
                icon="icons/refresh.png"/>
    </buttonsPanel>
    <rowsCount/>
    <columns>
        <column id="name"/>
        <column id="code"/>
    </columns>
    <rows datasource="budgetItemsDs"/>
</table>
</vbox>
</layout>
</window>

```

В элементе *dsContext* добавлен один компонент *источника данных collectionDatasource* , который выбирает сущности *BudgetItem* с помощью JPQL запроса select d from ext\$BudgetItem d order by d.name (Подробнее о JPQL можно прочитать на сайте <http://openjpa.apache.org/> ) с *представлением view="\_local"* .

Рассмотрим элементы дескриптора более подробно.

Компонент *Generic Filter* (XML-имя компонента – *filter* ) служит для отображения произвольного фильтра.



**Рисунок 50. Вид компонента Generic Filter**

Компонент *Table* (XML-имя компонента – *table* ) служит для отображения данных в виде сетки, упорядоченной в строки и столбцы.

1 строка	
Наименование	Код
Хозяйственные расходы	001

Рисунок 51. Вид компонента Table

Элемент `colItem` задает опции для колонки таблицы. Обязательно содержит атрибут `id`, в котором задается название атрибута сущности, выводимого в колонке. Название берется из *пакета локализованных сообщений*.

Для отображения количества строк таблицы используется элемент `rowCount`.

Над таблицей находится панель, содержащая кнопки для управления данными в этой таблице. XML-имя такой панели – `buttonsPanel`.



Рисунок 52. Вид компонента buttonsPanel

Размещение перечисленных выше компонентов задается с помощью контейнера `BoxLayout`. Существует три типа этого контейнера, определяемых именем XML-элемента. В нашем дескрипторе используется `vbox`, в результате чего элементы расположены вертикально.

*Дескрипторы экранов* имеют идентификаторы, по которым удобно вызывать экраны из меню или из программного кода. Идентификаторы назначаются в файле `ext-web-screens.xml` модуля `web`.

```
<screen id="ext$BudgetItem.browse"
template="/com/haulmont/ext/web/ui/budgetitem/dudgetitem-browse.xml"/>
```

Чтобы добавить экран в меню, измените файл `ext-web-menu.xml` модуля `web` следующим образом:

```
<menu-config>
...
<menu id="ext">
...
<item id="ext$BudgetItem.browse"/>
</menu>
</menu-config>
```

Если Вы хотите добавить новый пункт меню в уже существующее меню,

скопируйте содержимое файла thesis-web-menu.xml в файл ext-web-menu.xml и добавляйте созданный Вами пункт меню в существующую структуру. Например, чтобы добавить подпункт меню "**Статьи бюджета**" в меню "**Справочники**", измените файл ext-web-menu.xml следующим образом:

```
<menu-config xmlns="http://www.haulmont.com/schema/cuba/gui/menu-config.xsd">
    ...
    <menu id="reference">
        <!-- Taskman -->
        ...
        <!-- Docflow -->
        ...

        <!-- Extension -->
        <item id="ext$BudgetItem.browse"/>
    </menu>
    ...
</menu-config>
```

Для того чтобы ссылки отображались в меню корректно, создайте файлы **главного пакета сообщений расширения** messages.properties и messages\_ru.properties в пакете com.haulmont.ext.web модуля web , прописав в них названия пункта меню на английском и русском языках соответственно.

```
menu-config.ext$BudgetItem.browse=Статьи бюджета
```

Для локализации сообщений экрана создайте файлы messages.properties и messages\_ru.properties в пакете экрана com.haulmont.ext.web.ui.budgetitem . Например, для русской локализации файл будет иметь следующий вид:

```
browserCaption=Статьи бюджета
```

## 2. Класс контроллера будет называться BudgetItemBrowser . Описание класса представлено ниже.

```
package com.haulmont.ext.web.ui.budgetitem;

import com.haulmont.cuba.gui.ComponentsHelper;
import com.haulmont.cuba.gui.components.AbstractLookup;
import com.haulmont.cuba.gui.components.IFrame;
import com.haulmont.cuba.gui.components.Table;
import com.haulmont.cuba.gui.components.actions.RefreshAction;
import com.haulmont.cuba.web.filestorage.WebExportDisplay;
import com.haulmont.cuba.web.gui.components.WebFilter;
import com.haulmont.thesis.web.actions.ThesisExcelAction;

import javax.inject.Inject;
import java.util.Map;
```

```
public class BudgetItemBrowser extends AbstractLookup {

    @Inject
    protected Table budgetItemsTable;

    @Inject
    protected WebFilter genericFilter;

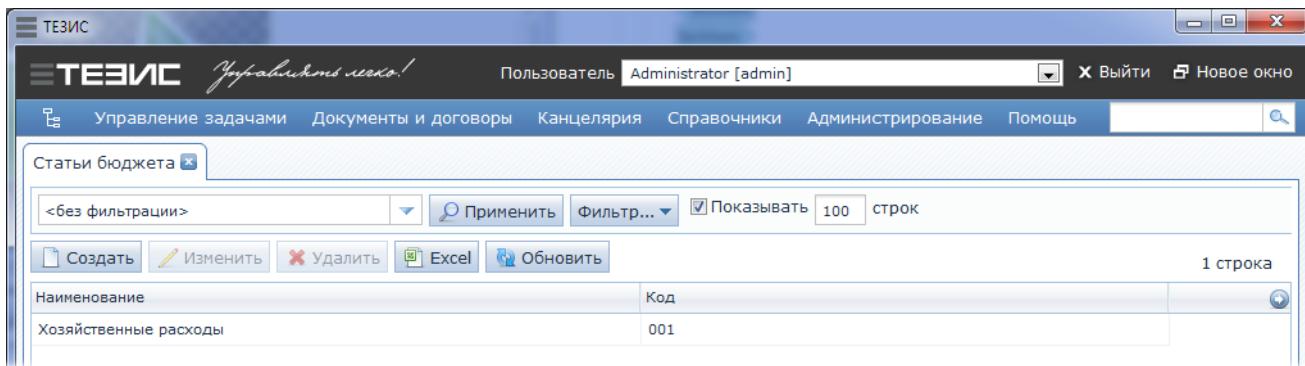
    public BudgetItemBrowser(IFrame frame) {
        super(frame);
    }

    @Override
    public void init(Map<String, Object> params) {
        super.init(params);
        ComponentsHelper.createActions(budgetItemsTable);
        budgetItemsTable.addAction(new RefreshAction(budgetItemsTable));
        budgetItemsTable.addAction(new ThesisExcelAction(budgetItemsTable,
            new WebExportDisplay(), genericFilter));
    }
}
```

BudgetItemBrowser наследуется от AbstractLookup и переопределяет метод init(Map<String, Object> params), который вызывается при формировании окна. Переопределение метода необходимо для того, чтобы создать обработчики событий на кнопки, объявленные в budgetitem-browse.xml. Так как в большинстве списков необходимы стандартные операции (создание, изменение, удаление), воспользуемся классом ComponentsHelper со статическим методом createActions в который передается компонент таблицы, к которому привязаны действия.

Доступ к компонентам экрана осуществляется с помощью механизма [внедрения зависимости](#). Работать с компонентами нужно через интерфейсы, объявленные в пакете com.haulmont.cuba.gui.components.

Посмотрим, как созданный нами экран выглядит в системе. Для этого пересоберите проект и зайдите в систему. Откройте пункт меню **Справочники -> Статьи бюджета**.



**Рисунок 53. Окно управления статьями бюджета**

Можно убедиться, что данный экран содержит следующие элементы:

- Панель с фильтром.
- Панель с кнопками **Создать**, **Изменить**, **Удалить**, **Обновить**, **Excel** и с меткой, отображающей число записей в таблице.
- Таблицу с колонками "**Код**" и "**Наименование**".

### Создание окна редактирования сущности

Экран редактирования сущности "Статья бюджета" состоит из:

1. **XML-дескриптора** `budgetitem-edit.xml`. Для того чтобы экран редактирования сущности "Статья бюджета" имел поля для ввода наименования и кода, XML-дескриптор должен иметь следующий вид:

```

<window
    xmlns = "http://schemas.haulmont.com/cuba/5.2/window.xsd"
    class = "com.haulmont.ext.web.ui.budgetitem.BudgetItemEditor"
    caption = "msg://editorCaption"
    datasource = "budgetItemDs"
    messagesPack = "com.haulmont.ext.web.ui.budgetitem"
    focusComponent = "fieldGroup">

    <dsContext>
        <datasource id = "budgetItemDs"
            class = "com.haulmont.ext.core.entity.BudgetItem"
            view = "_local"/>
    </dsContext>

    <layout expand = "windowActions">
        <vbox spacing = "true">
            <fieldGroup id = "fieldGroup"
                datasource = "budgetItemDs"
                stylename = "edit-area">
                <column width = "250px">
                    <field id = "name"
                        required = "true"/>
                    <field id = "code"/>
                </column>
            </fieldGroup>
        </vbox>
    </layout>

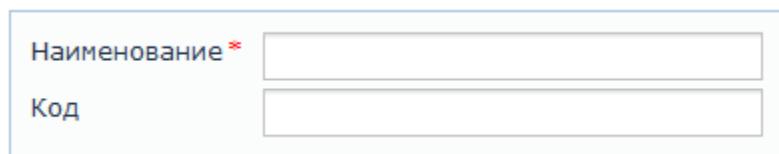
```

```

        </column>
        </fieldGroup>
    </vbox>
    <iframe id = "windowActions"
        src = "/com/haulmont/cuba/gui/edit-window.actions.xml"/>
    </layout>
</window>
    
```

В экран редактирования будет передаваться одна сущность либо на создание, либо на изменение и проставляться в *datasource*. Поэтому вместо *collectionDatasource* объявлен *datasource*.

Для отображения набора атрибутов редактируемой или создаваемой сущности используется компонент **FieldGroup** (XML-имя компонента – *fieldGroup* ).



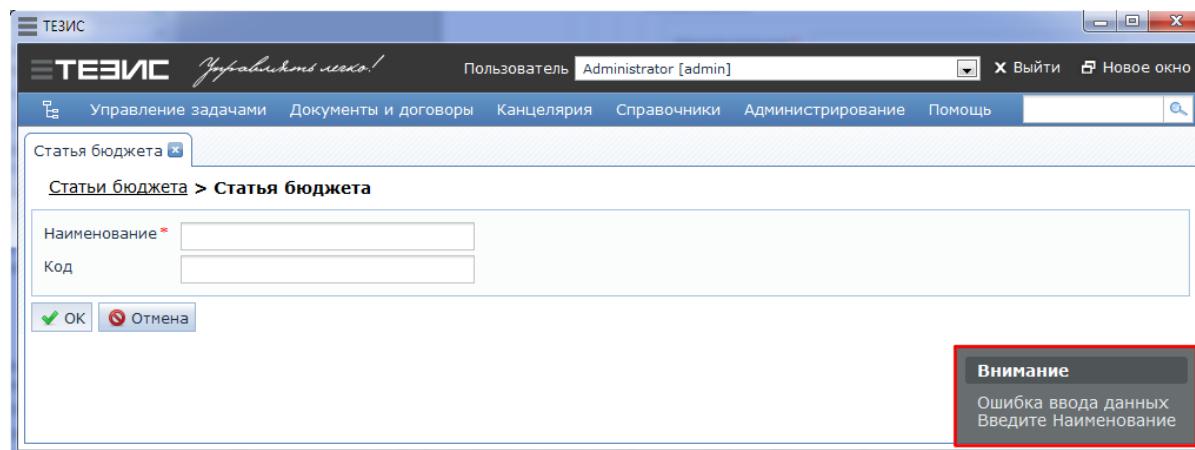
**Рисунок 54. Вид компонента FieldGroup**

Данный компонент содержит в себе два элемента *field* :

```

<field id = "name"
    required = "true"/>
<field id = "code"/>
    
```

Обратите внимание, что в первом элементе *field* указан атрибут *required = "true"*. Данный атрибут указывает, что в данное поле обязательно должно быть введено значение. Такое поле помечается красной звездочкой. В случае если значение в данное поле не было введено, пользователю выводится сообщение.



**Рисунок 55.**

*XML-дескриптор* экрана редактирования должен содержать фрейм с кнопками подтверждения или отказа операции над сущностью. Есть два стандартных варианта такого фрейма:

1. /com/haulmont/cuba/gui/edit-window.actions.xml – фрейм с двумя кнопками: **OK**, **Cancel**. При нажатии на кнопку **OK** происходит фиксация операции, и окно закрывается.
2. /com/haulmont/cuba/gui/extended-edit-window.actions.xml – фрейм с тремя кнопками: **OK**, **OK&Close**, **Cancel**. В этом фрейме есть возможность подтвердить операцию, не закрывая окна.

Зарегистрируйте экран в файле ext-web-screens.xml модуля web , чтобы его можно было открывать из меню, других окон или из программного кода.

```
<screen id="ext$BudgetItem.edit"
    template="/com/haulmont/ext/web/ui/budgetitem/budgetitem-edit.xml"/>
```

Для корректного отображения название редактируемой сущности нужно добавить *локализованное название* сущности в файлы messages.properties и messages\_ru.properties в пакете com.haulmont.ext.web.ui.budgetitem . Например, для русской локализации в файл будет добавлена следующая строка:

```
editorCaption=Статья бюджета
```

2. Класс контроллера для редактирования сущности должен быть унаследован от com.haulmont.cuba.gui.components.AbstractEditor , который реализует интерфейс Window.Editor .

В контроллере часто приходится переопределять метод setItem() , так как в этот момент становится известным экземпляр, который будет редактироваться. В зависимости от свойств этого экземпляра может быть произведена та или иная инициализация. Метод setItem() вызывается после init() .

Для перехвата момента фиксации операции и закрытия окна необходимо в контроллере переопределить метод commitAndClose() . В качестве примера рассмотрим случай, когда перед фиксацией изменений нужно проверить, не пусто ли поле с наименованием статьи бюджета. Класс контроллера будет иметь следующий вид:

```
package com.haulmont.ext.web.ui.budgetitem;
```

```
import com.haulmont.cuba.core.entity.Entity;
import com.haulmont.cuba.gui.components.AbstractEditor;
import com.haulmont.cuba.gui.components.IFrame;
import com.haulmont.cuba.gui.data.Datasource;
import com.haulmont.ext.core.entity.BudgetItem;

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

public class BudgetItemEditor extends AbstractEditor {

    @Inject
    private Datasource<BudgetItem> budgetItemDs;

    public BudgetItemEditor(IFrame frame) {
        super(frame);
    }

    @Override
    public void setItem(Entity item) {
        super.setItem(item);
    }

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

    @Override
    public void commitAndClose() {
        if ((budgetItemDs.getItem()).getName() == null) {
            showNotification(getMessage("dialogMessage"),
                NotificationType.HUMANIZED);
        } else {
            super.commitAndClose();
        }
    }
}
```

Обратите внимание, в классе контроллера был использован механизм **внедрения зависимости** (англ. Dependency injection ) для получения ссылок на используемые объекты.

```
@Inject
private Datasource budgetItemDs;
```

Согласно этому механизму в текущем экране будет произведен поиск **источника данных** с соответствующим именем.

Если в окне редактирования возможна промежуточная фиксация операции без закрытия окна, переопределить нужно два метода: `commitAndClose()` и `commit()`.

Если требуется изменить содержимое сущности на этапе фиксации ее в БД, а с окном ничего делать не нужно, рекомендуется пользоваться слушателем

DsContext.CommitListener .

```
package com.haulmont.ext.web.ui.budgetitem;

import com.haulmont.cuba.core.entity.Entity;
import com.haulmont.cuba.core.global.CommitContext;
import com.haulmont.cuba.gui.components.AbstractEditor;
import com.haulmont.cuba.gui.components.IFrame;
import com.haulmont.cuba.gui.data.Datasource;
import com.haulmont.ext.core.entity.BudgetItem;
import com.haulmont.cuba.gui.data.DsContext;

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

public class BudgetItemEditor extends AbstractEditor {

    @Inject
    private Datasource<BudgetItem> budgetItemDs;

    public BudgetItemEditor(IFrame frame) {
        super(frame);
    }

    @Override
    public void setItem(Entity item) {
        super.setItem(item);
    }

    @Override
    public void init(Map<String, Object> params) {
        super.init(params);
        getDsContext().addListener(new DsContext.CommitListener() {
            public void beforeCommit(CommitContext<Entity> context) {
                if ((budgetItemDs.getItem()).getName() == null) {
                    (budgetItemDs.getItem()).setName("default name");
                    showNotification(getMessage("notificationMessage"),
                        NotificationType.HUMANIZED);
                }
            }

            public void afterCommit(CommitContext<Entity> context,
                                   Map<Entity, Entity> result) {
            }
        });
    }
}
```

## 4.2. Создание новой карточки

В данном разделе пошагово описывается создание карточки "Счет на оплату".

### 4.2.1. Создание таблицы "Счет на оплату"

Любая карточка, которая будет привязана к процессу , должна создаваться по следующему принципу.

Базовой таблицей для всех карточек является **WF\_CARD**. Все дочерние карточки наследуются от этой таблицы путем создания новой таблицы со ссылкой на запись в **WF\_CARD**. То есть один объект системы состоит из записи в **WF\_CARD**, где хранятся системные поля, и записей в наследуемых таблицах.

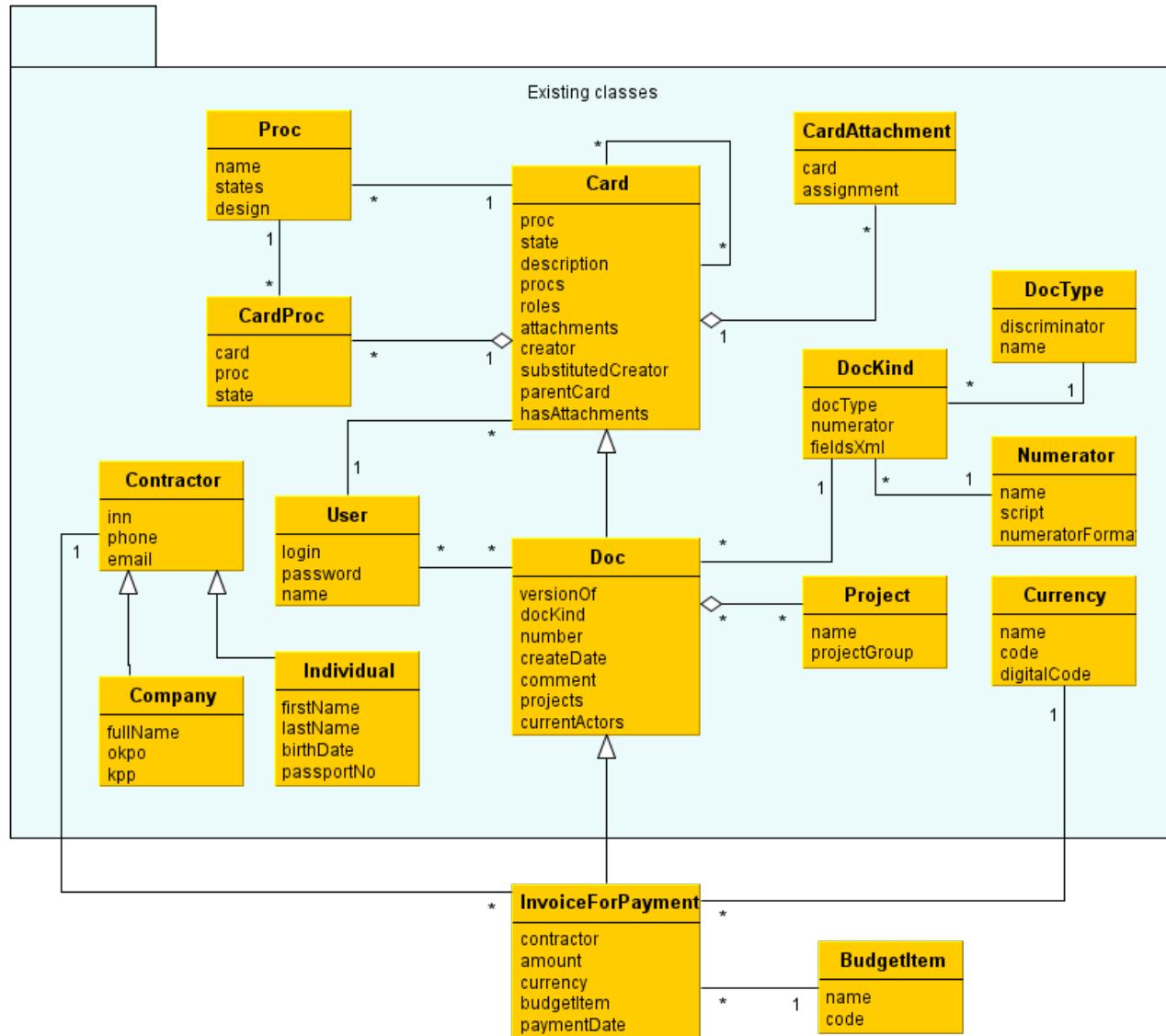
Создадим таблицу **EXT\_INVOICE\_FOR\_PAYMENT**.

```

create table EXT_INVOICE_FOR_PAYMENT (
    CARD_ID uuid,
    BUDGET_ITEM_ID uuid,          --Статья бюджета
    CONTRACTOR_ID uuid,          --Контрагент
    PAYMENT_DATE timestamp,       --Дата оплаты
    AMOUNT numeric(19,2),         --Сумма
    CURRENCY_ID uuid,            --Валюта
    primary key (CARD_ID)
) ^
-- Связь счета на оплату с DF_DOC осуществляется по полю CARD_ID
alter table EXT_INVOICE_FOR_PAYMENT
add constraint FK_EXT_INVOICE_FOR_PAYMENT_DOC
foreign key (CARD_ID) references DF_DOC(CARD_ID)
^
alter table EXT_INVOICE_FOR_PAYMENT
add constraint FK_EXT_INVOICE_FOR_PAYMENT_BUDGET_ITEM
foreign key (BUDGET_ITEM_ID) references EXT_BUDGET_ITEM(ID)
^
alter table EXT_INVOICE_FOR_PAYMENT
add constraint FK_EXT_INVOICE_FOR_PAYMENT_CURRENCY
foreign key (CURRENCY_ID) references DF_CURRENCY(ID)
^
alter table EXT_INVOICE_FOR_PAYMENT
add constraint FK_EXT_INVOICE_FOR_PAYMENT_CONTRACTOR
foreign key (CONTRACTOR_ID) references DF_CONTRACTOR(CORRESPONDENT_ID)
^
  
```

Так как счет на оплату является разновидностью документа, сущность Счет на оплату наследуется от сущности Документ, а таблица **EXT\_INVOICE\_FOR\_PAYMENT** содержит ссылку на таблицу **DF\_DOC**. Таблица **DF\_DOC** содержит ссылку на таблицу **WF\_CARD**.

Далее приведена диаграмма классов, частично отражающая предметную область.



**Рисунок 56. Диаграмма классов**

#### 4.2.2. Создание класса сущности `InvoiceForPayment`

Классы сущности должны быть унаследованы от класса `com.haulmont.workflow.core.entity.Card`, который содержит в себе все системные поля для участия карточки в *процессе*. Так как счет на оплату является разновидностью документа, унаследуйте класс `InvoiceForPayment` от класса `Doc`, который, в свою очередь, унаследован от класса `Card`. В результате класс `InvoiceForPayment` будет выглядеть следующим образом:

```

package com.haulmont.ext.core.entity;

import com.haulmont.chile.core.annotations.NamePattern;
import
com.haulmont.cuba.core.entity.annotation.Listenerscom.haulmont.thesis.core.entityre.entity.Contracto
com.haulmont.thesis.core.entity.Currency;
import com.haulmont.thesis.core.entity.Doc;

import javax.persistence.*;
import java.math.BigDecimal;
import java.util.Date;

@Entity(name = "ext$InvoiceForPayment")
@Table(name = "EXT_INVOICE_FOR_PAYMENT")
@DiscriminatorValue("130")
@PrimaryKeyJoinColumn(name = "CARD_ID", referencedColumnName =
"CARD_ID")
@Listeners("com.haulmont.thesis.core.listener.DocEntityListener")
@NamePattern("%s|description")

public class InvoiceForPayment extends Doc {
    private static final long serialVersionUID = -5772794459098915083L;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "CONTRACTOR_ID")
    protected Contractor contractor;

    @Column(name = "AMOUNT")
    private BigDecimal amount;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "CURRENCY_ID")
    protected Currency currency;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "BUDGET_ITEM_ID")
    protected BudgetItem budgetItem;

    @Column(name = "PAYMENT_DATE")
    //помечаем атрибут date, что он должен хранить дату без времени
    @Temporal(TemporalType.DATE)
    private Date paymentDate;

    //Методы доступа
    ...
}
  
```

В этом классе следует обратить внимание на следующие аннотации:

- `@DiscriminatorValue`. `@DiscriminatorValue` хранится в корневой таблице **WF\_CARD**. По этому значению для каждой строки таблицы **WF\_CARD** можно определить, к какой карточке (Документ, Задача, Договор или иная карточка) относится эта строка.
- `@PrimaryKeyJoinColumn`. Эта аннотация используется для указания полей для связи с базовой таблицей. В нашем примере сущность Счет на оплату наследуется от сущности Документ, а таблица

**EXT\_INVOICE\_FOR\_PAYMENT** ссылается на таблицу **DF\_DOC** по первичному ключу **CARD\_ID**, который соответствует колонке **CARD\_ID** таблицы **DF\_DOC**.

После создания класса его нужно зарегистрировать в файле `ext-persistence.xml` модуля `global`:

```
<class>com.haulmont.ext.core.entity.InvoiceForPayment</class>
```

Для локализации названий свойств сущности необходимо добавить перевод в файлы `messages*.properties` пакета `com.haulmont.ext.core.entity` модуля `global`.

Например, для русской локализации `InvoiceForPayment` перевод будет следующим:

```
InvoiceForPayment=Счет на оплату
InvoiceForPayment.budgetItem=Статья бюджета
InvoiceForPayment.contractor=Контрагент
InvoiceForPayment.parentCard=Основание
InvoiceForPayment.paymentDate=Дата оплаты
InvoiceForPayment.number=Номер счета
InvoiceForPayment.amount=Сумма
InvoiceForPayment.description=Описание
```

### 4.2.3. Создание экранов

Файлы, относящиеся к экранам, необходимо создавать в пакете `com.haulmont.ext.web.ui`

В первую очередь создайте папку `invoiceforpayment`, где мы будем создавать файлы, относящиеся к экранам сущности `InvoiceForPayment`.

#### Создание списка карточек

Для создания списка "Счета на оплату" необходимо создать следующие файлы:

1. XML-дескриптор списка сущностей `invoiceforpayment-browse.xml`.

Определимся, какие элементы должны быть на экране, отображающем список счетов на оплату.

1. Таблица, содержащая карточки типа "Счет на оплату", с колонками **Статья бюджета**, **Контрагент**, **Основание**, **Дата оплаты**, **Сумма**, **Валюта**, **Номер**, **Описание платежа**, **Текущий процесс**, **Текущее состояние**. Таблица должна иметь возможность динамической группировки по любому полю.

2. Фильтр для поиска по заданным условиям.

3. Кнопка **Создать**, позволяющая создать счет на оплату двумя

способами, открывающимися в выпадающем списке, а именно: создать новый счет на оплату с помощью кнопки **Новый** и создать на основе копирования существующего счета на оплату с помощью кнопки **Копировать**.

4. Кнопки **Изменить**, **Удалить**, **Обновить**, **Excel**.
5. Флажок **Скрыть журнал действий**, определяющий, отображается ли для выбранной задачи журнал действий (в нижней части экрана).
6. Текстовое поле, содержащее информацию о количестве карточек в таблице.

Для того чтобы в таблицу загружались карточки счетов на оплату, необходимо подключить к этой таблице *источник данных*. В качестве источника данных используется **groupDatasource**, который позволяет динамически группировать данные в **groupTable** по любому атрибуту.

```
&lt;dsContext&ampgt
  &lt;groupDatasource id="invoicesForPaymentDs";
    class="com.haulmont.ext.core.entity.InvoiceForPayment";
    view="browse"&gt;

  &lt;datasourceClass&ampgtcom.haulmont.thesis.web.ui.DocDatasource&lt;/datasourceClass&ampgt;
  &lt;query&ampgt
    &lt;! [CDATA[select distinct i from ext$InvoiceForPayment i
      where i.id &lt;:param$exclItem
      and i.template = false
      and i.versionOf is null
      order by i.dateTime]]&gt;
  &lt;/query&ampgt
  &lt;/groupDatasource&gt;
&lt;/dsContext&ampgt
```

Для реализации фильтра в дескрипторе списка сущностей напишите следующее:

```
<filter id="genericFilter"
  datasource="invoicesForPaymentDs"
  stylename="edit-area">
  <properties include=".*"
    exclude="template|templateName|createdBy
      |regNo|regDate|id|jbpmProcessId
      |createTs|docOfficeData|docKind
      |docCategory|category|locState
      |description|state|"/>
</filter>
```

Компонент **Generic Filter** (XML-имя компонента – **filter**) служит для создания и применения во время работы приложения произвольных фильтров для *источников данных*.

Кнопка, позволяющая создать счет на оплату двумя способами, кнопки **Изменить**, **Удалить**, **Обновить**, **Excel**, а также опция **Скрыть журнал действий** должны быть расположены в компоненте **ButtonsPanel** :

```
<buttonsPanel>
    <popupButton id="createButton"
        icon="icons/create.png"
        caption="msg://actions.Create"
        width="${100}"
        menuWidth="${115}"/>
    <button action="invoicesForPaymentTable.edit"
        icon="icons/edit.png"/>
    <button action="invoicesForPaymentTable.remove"
        icon="icons/remove.png"/>
    <button action="invoicesForPaymentTable.refresh"
        icon="icons/refresh.png"/>
    <button action="invoicesForPaymentTable.excel"
        icon="icons/excel.png"/>
    <checkBox id="hideResolutions"
        caption="msg://hideResolutions"
        value="true" />
</buttonsPanel>
```



**Рисунок 57. Вид компонента ButtonsPanel**

Для того чтобы отображать информацию о количестве карточек в таблице, необходимо после компонента **ButtonsPanel** добавить элемент **RowCount**:

```
<groupTable id="invoicesForPaymentTable"
    editable="false">
    <buttonsPanel>
        ...
    </buttonsPanel>
    <rowCount/>
    ...
    <rows datasource="invoicesForPaymentDs"/>
</groupTable>
```

Определим **представления** в файле `ext-views.xml` модуля `core`.

### Внимание

Исключение для задания имен представлений: так как карточка счета на оплату унаследована от карточки документа, существуют некоторые ограничения на именование представлений. В данном случае представление можно назвать только "browse".

```
<view class="com.haulmont.ext.core.entity.InvoiceForPayment"
      name="browse"
      extends="_local">
<property name="contractor"
          view="_minimal"/>
<property name="cuurrency"
          view="_minimal"/>
<property name="budgetItem"
          view="_minimal"/>
<property name="proc"
          view="_local"/>
</view>
```

Далее необходимо зарегистрировать экран в файле `ext-web-screens.xml` модуля `web`, чтобы его можно было открывать из меню, других окон или из программного кода.

```
<screen id="ext$InvoiceForPayment.browse"
       template="/com/haulmont/ext/web/ui/invoiceforpayment/invoiceforpayment-browse.xml"/>
```

Меню сделаем по аналогии с меню документов. В пункте **Документы и договоры** добавим подпункт **Создать счет на оплату** и подпункт **Счета на оплату**. Для этого нужно добавить в файл `ext-web-menu.xml` модуля `web` следующее:

```
<menu id="docflow">
    <item id="ext$InvoiceForPayment.edit">
        <permissions>
            <permission type="ENTITY_OP" target="ext$InvoiceForPayment:create"/>
        </permissions>
    </item>
    <item id="ext$InvoiceForPayment.browse"/>
    <item id="-"/>
    ...
</menu>
```

Для того чтобы ссылки в меню отображались корректно, необходимо в **локализованные файлы** `messages*.properties` в пакете `com.haulmont.ext.web.ui` модуля `web` добавить названия пунктов меню. Например, для русского языка следует добавить следующую строку:

```
menu-config.ext$InvoiceForPayment.browse=Счета на оплату
menu-config.ext$InvoiceForPayment.edit=Создать счет на оплату
```

Для корректного отображения имени списка сущностей нужно создать файлы `messages*.properties` в пакете `com.haulmont.ext.web.ui.invoiceforpayment` модуля `web`. Например, для русской локализации файл будет иметь следующий вид:

```
browserCaption=Счета на оплату
```

2. Класс контроллера будет называться `InvoiceForPaymentBrowser`.

Класс `InvoiceForPaymentBrowser` наследуется от класса `AbstractLookup` и переопределяет метод `init(Map<String, Object> params)`, который вызывается при формировании окна.

Рассмотрим метод `init()` более подробно. Переопределение метода необходимо для того, чтобы создать обработчики событий на кнопки Создания, Изменения, Удаления, Обновления и Выгрузки в Excel. Также в этом методе добавляется слушатель `CollectionDsListenerAdapter`, который срабатывает при обновлении источника данных `groupDatasource`. Это необходимо для подгрузки списка уведомлений `infolist`, чтобы показывать строчку в таблице жирным, если на данную карточку у пользователя имеется уведомление.

В методе `init()` также устанавливается возможность одновременного выбора нескольких карточек:

```
invoicesForPaymentTable.setMultiSelect(true);
```

Далее рассмотрим, как создается кнопка **Создать** с выпадающим списком действий по созданию карточки счета на оплату.

```
public void init(Map<String, Object> params) {
    /*...*/
    if (createButton != null) {
        //проверка прав текущего пользователя на создание карточек типа
        //Счет на оплату. В зависимости от прав пользователя установка
        //возможности создавать карточку
        createButton.setEnabled(UserSessionProvider.getUserSession()
            .isEntityOpPermitted(invoicesForPaymentDs.getMetaClass(),
                EntityOp.CREATE));
        //добавление действия "Создать новую карточку"
        createButton.addAction(new CreateAction(invoicesForPaymentTable) {
            //получение названий для данной кнопки из локализованного файла
            //сообщений
            @Override
            public String getCaption() {
                final String messagesPackage = getMessagesPack();
                return MessageProvider.getMessage(messagesPackage,
                    "actions.createNew");
            }
        });
        //добавление действия "Копировать существующую карточку"
        createButton.addAction(new CopyAction());
    }
    /*...*/
}
```

В обработчике событий `CopyAction()` создается новая карточка типа Счет на оплату, и в поля этой карточки с помощью методов доступа заносятся значения из выбранной пользователем карточки.

Также необходимо создать обработчик событий `EditAction()`. В обработчике нужно указать, что при закрытии окна редактирования таблица счетов на оплату должна обновляться. Такое обновление таблицы необходимо в процессе обмена комментариями по данному счету на оплату, а также при просмотре уведомлений по процессу, запущенному для данного счета на оплату.

Кроме действий по удалению, обновлению, изменению и выгрузки в Excel необходимо добавить действие по удалению просмотренных уведомлений:

```
invoicesForPaymentTable.addAction(new DeleteNotification());
```

Многострочное описание должно отображаться в таблице карточек как ссылка, при нажатии которой должно отобразиться всплывающее окно, отображающее многострочное описание. Для этого напишем метод `addCommentColumn()`, вызов которого происходит в методе `init()`.

```
protected void addCommentColumn() {
    MetaPropertyPath pp =
    invoicesForPaymentDs.getMetaClass().getPropertyPath("comment");
    com.vaadin.ui.Table vTable = (com.vaadin.ui.Table) WebComponentsHelper
        .unwrap(invoicesForPaymentTable);
    vTable.removeGeneratedColumn(pp);
    vTable.addGeneratedColumn(
        pp,
        new com.vaadin.ui.Table.ColumnGenerator() {
            public com.vaadin.ui.Component generateCell(com.vaadin.ui.Table
source,
                                               Object itemId, Object
columnId) {
                UUID uuid = (UUID) itemId;
                com.vaadin.ui.Component component;
                Doc doc = invoicesForPaymentDs.getItem(uuid);
                String descr = doc.getComment();
                int enterIdx = descr != null ?
                    (descr.length() > 40 ? 40 : descr.indexOf('\n')) :
                    -1;
                //установка свойств отображаемого компонента
                if (enterIdx != -1) {
                    com.vaadin.ui.TextField content = new
com.vaadin.ui.TextField(null,
                descr);
                    content.setReadOnly(true);
                    content.setWidth("300px");
                    content.setHeight("300px");
                    component = new com.vaadin.ui.PopupView("<span>" +
                        descr.substring(0, enterIdx) + "...</span>", content);
                    component.addStyleName("longtext");
                } else {
                    component = new com.vaadin.ui.Label(descr == null ? "" :
descr);
                }
                component.setWidth("-1px");
            }
        });
}
```

```
        return component;
    }
});
```

В данном методе для задания собственного представления данных в колонке **comment** необходимо добавить генерируемую колонку с помощью метода `Table.addGeneratedColumn()`.

В таблице существует колонка **Текущее состояние**, в которой отображается название состояния, в котором находится карточка в запущенном процессе. Добавим возможность изменения состояния непосредственно в списке просмотра. Для этого создайте метод `initAfterSetLookup(boolean isLookup)`:

```
private void initAfterSetLookup(boolean isLookup) {
    if (isLookup) {
        /*...*/
    }
    //иначе добавляется генерируемая колонка
    else {
        ((com.vaadin.ui.Table) WebComponentsHelper
            .unwrap(invoicesForPaymentTable))
            .addGeneratedColumn(invoicesForPaymentDs.getMetaClass()
                .getPropertyPath("locState"),
                new com.vaadin.ui.ColumnGenerator() {
                    public com.vaadin.ui.Component
                        generateCell(com.vaadin.ui.Table source,
                            Object itemId, Object columnId) {
                        UUID uuid = (UUID) itemId;
                        com.vaadin.ui.Component component;
                        Doc doc = invoicesForPaymentDs.getItem(uuid);

                        boolean containsCard = cards != null ?
                            cards.contains(doc) : false;
                        if (doc != null && !WfUtils.isCardInStateList(doc,
                            "Finished", "Canceled") &&
                            (containsCard || (!containsCard && WfUtils
                                .isCardInState(doc, "New")))) {
                            //формируется кнопка, при нажатии на которую
                            //отображается выпадающий список с доступными
                            //для данной карточки действиями
                            final PopupButton popupButton = new WebPopupButton();
                            //получение списка состояний
                            popupButton.setCaption(invoicesForPaymentDs
                                .getItem((UUID) itemId).getLocState());
                            popupButton.addAction(new AbstractAction("onChange") {
                                public void actionPerformed(Component component) {
                                    if (invoicesForPaymentTable
                                        .getSelected().size() > 1) {
                                        popupButton.setEnabled(false);
                                    } else
                                        popupButton.setEnabled(true);
                                }
                            });
                            component = WebComponentsHelper.unwrap(popupButton);
                            ((org.vaadin.hene.popupbutton.PopupButton) component)
```

```
        .addListener(new ProcessMenuBuilderAction(doc,
            invoicesForPaymentTable, popupButton));
        component.addStyleName("link");
        component.addStyleName("dashed");
    } else {
        component = new com.vaadin.ui.Label(doc == null ? "" :
            doc.getLocState(),
            com.vaadin.ui.Label.CONTENT_XHTML);
    }
    component.setWidth("-1px");
    return component;
}
/*...
}
}
```

В этом методе также используется генерируемая колонка для задания собственного представления данных в виде ссылки, при нажатии которой отображается выпадающий список с доступными для данной карточки действиями.

### Совет

Для более подробного описания XML-дескриптора и класса контроллера обращайтесь к исходным кодам проекта.

## Создание окна редактирования карточки

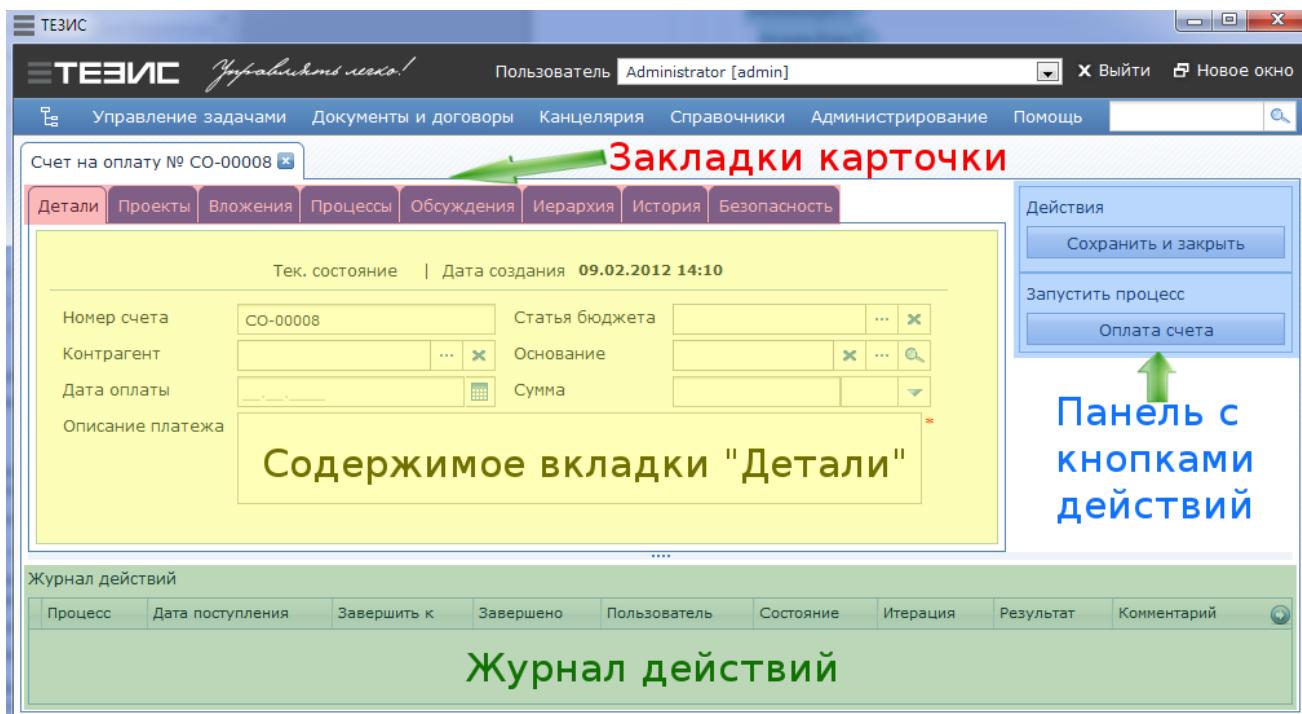
Определимся, как должно выглядеть окно редактирования карточки "Счет на оплату".

- Карточка должна иметь следующие вкладки:
  - Детали
  - Проекты
  - Вложения
  - Процессы
  - Обсуждения
  - Иерархия
  - История
  - Безопасность

Подробное описание, что должна содержать каждая из этих вкладок,

содержится в Главе 4.2 *Руководства пользователя системы ТЕЗИС*.

- Вкладка **Детали** должна иметь следующие поля:
  - Номер счета – текстовое поле.
  - Контрагент – поле с возможностью создания нового или выбора уже существующего контрагента (как юридического лица, так и физического).
  - Дата оплаты – поле с возможностью выбора даты из календаря.
  - Статья бюджета – поле с возможностью создания новой или выбора уже существующей статьи бюджета.
  - Основание – поле с возможностью выбора карточки, являющейся "родительской" для текущей карточки.
  - Сумма – текстовое поле для ввода суммы оплаты данного счета.
  - Валюта – выпадающий список доступных валют, в которых указывается сумма оплаты данного счета.
  - Описание – текстовое поле для дополнительной информации.
- Внизу экрана должен отображаться журнал действий по процессу.



**Рисунок 58. Окно редактирования карточки счета на оплату**

Создайте XML-дескриптор `invoiceforpayment-edit.xml`.

Особого внимания заслуживает элемент для инициализации источников данных:

```
<?dsContext?>
<datasource id="cardDs"
    class="com.haulmont.ext.core.entity.InvoiceForPayment";
    view="edit">
<collectionDatasource id="cardProcDs"
    property="procs"/>
<collectionDatasource id="cardRolesDs"
    property="roles"/>
<collectionDatasource id="cardProjectsDs"
    property="projects"/>
</datasource>

<collectionDatasource id="currencyDs"
    class="com.haulmont.thesis.core.entity.Currency";
    view="_local"/>
<!--...-->

<hierarchicalDatasource id="attachmentsDs"
    class="com.haulmont.workflow.core.entity.CardAttachment";
    view="card-edit";
    hierarchyProperty="versionOf">
    <query>select a from wf$CardAttachment a
        where a.card.id = :ds$cardDs order by a.createTs desc
    </query>
</hierarchicalDatasource>
</dsContext>
```

Следует отметить, что внутри *источника данных* существуют вложенные источники данных. Вложенные источники нужны для работы с экземплярами связанных сущностей, загруженных вместе с основной. Описание вложенного источника всегда содержит атрибут *property*, указывающий на атрибут основной сущности. Также, помимо использования *CollectionDatasource* (например, *currencyDs*), возникла потребность использовать *HierarchicalDatasource* (*attachmentsDs*).

*HierarchicalDatasource* расширяет *CollectionDatasource* для поддержки иерархических визуальных компонентов.

В XML-дескрипторе присутствует элемент *accessControl*, который необходим во всех редакторах карточек и должен быть унаследован от класса com.haulmont.thesis.gui.processaction.DefaultCardAccessData. Это псевдокомпонент, с помощью которого можно управлять видимостью или обязательностью полей.

```
<accessControl  
data="com.haulmont.ext.web.ui.invoiceforpayment.InvoiceForPaymentAccessData"  
param="accessData"/>
```

Рассмотрим структуру XML-дескриптора:

```
<layout expand="split">  
    <split id="split"  
        orientation="vertical"  
        pos="70">  
        <hbox id="mainPane"  
            expand="tabsheet"  
            spacing="true">  
            <tabsheet id="tabsheet">  
                <tab id="mainTab"  
                    caption="msg://mainTab"  
                    margin="true">  
                    <groupBox stylename="edit-area">  
                        <scrollbox>  
                            <vbox spacing="true"  
                                width="-1px"  
                                height="-1px">  
                                <grid id="captionInfo"  
                                    spacing="true"  
                                    align="TOP_CENTER"  
                                    margin="true, false, false, false">  
                                    <!--...-->  
                                </grid>  
                                <grid stylename="separator"  
                                    margin="true"  
                                    spacing="true"  
                                    expandable="false">  
                                    <!--...-->  
                                </grid>  
                            </vbox>  
                        </scrollbox>  
                    </groupBox>  
                </tab>
```

```
<tab id="cardProjectsTab" caption="msg://cardProjectsTab" lazy="true">
    <iframe id="cardProjectsFrame"
        src="/com/haulmont/thesis/web/ui/common/card-projects-frame.xml"/>
    </tab>

    <!---->
</tabsheet>

<vbox margin="false,false,false,true"
      width="-1"
      spacing="true"
      height="100%">
    <!---->
</vbox>
</hbox>

<vbox id="resolutionsPane"
      expand="resolutionsFrame"
      height="250px"
      spacing="true">
    <label value="msg://resolutions"/>
    <iframe id="resolutionsFrame"
            src="/com/haulmont/workflow/web/ui/base/resolutions-frame.xml"/>
</vbox>
</split>
</layout>
```

В контейнере **Layout** установлен атрибут **expand="split"** . Это означает, что компоненту **split** будет установлена максимально возможная высота и ширина. Компонент **SplitPanel** (XML-имя компонента – **split** ) – это контейнер, разбитый на две области, размер которых по одному из направлений (горизонтали или вертикали) можно менять путем перемещения границы этих областей.

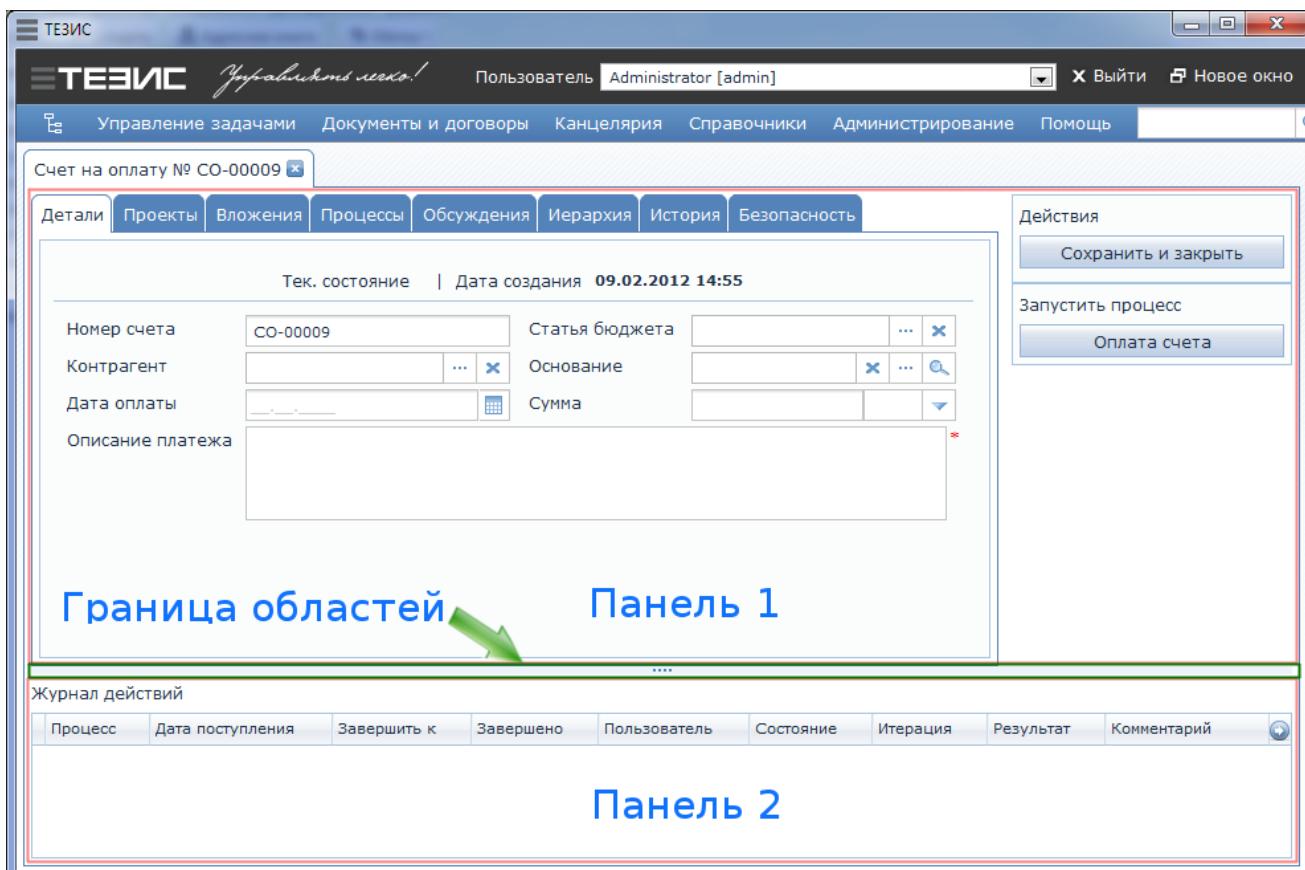


Рисунок 59. Окно редактирования карточки счета на оплату

*SplitPanel* в нижней области содержит контейнер *vbox id="resolutionsPane"*, который, в свою очередь, содержит фрейм журнала действий, а в верхней области – контейнер *hbox id="mainPane"*. Контейнер *mainPane* содержит компонент с вкладками (*tabsheet id="tabSheet"*), в которых отображается информация по карточке, а также контейнер, содержащий кнопки сохранения карточки, запуска процессов, доступных для данной карточки.

Рассмотрим вкладку **Детали** более подробно. Вкладка содержит в себе контейнер *GroupBoxLayout*, благодаря чему компоненты, располагаемые в этом контейнере, будут выделены рамкой. *GroupBoxLayout* содержит в себе специальный контейнер, позволяющий располагать объекты по сетке – *GridLayout*. Для того чтобы добавить поле **Контрагент** с возможностью создания нового или выбора уже существующего контрагента, воспользуемся компонентом *pickerField*:

```
<pickerField id="contractor"
    datasource="cardDs"
    property="contractor"
    width="${width}"
    lookupScreen="df$Contractor.lookup"/>
```

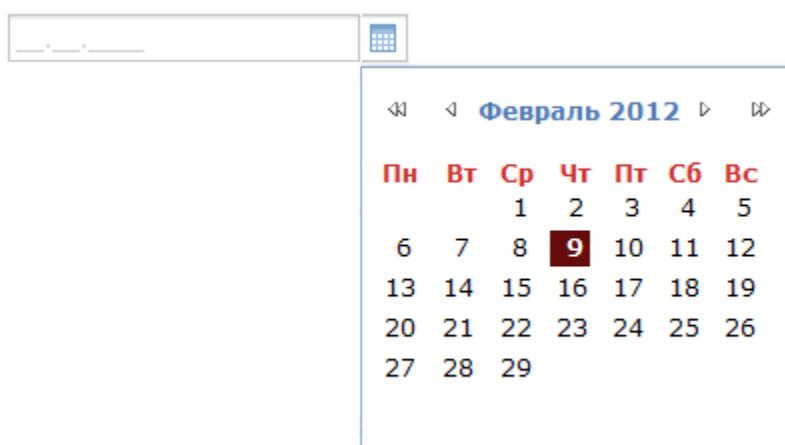


**Рисунок 60. Вид компонента pickerField**

По такому же принципу создайте поля **Статья бюджета** и **Основание**.

Для создания поля **Дата оплаты** воспользуемся компонентом *dateField* :

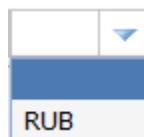
```
<dateField id="date"
           datasource="cardDs"
           property="paymentDate"
           width="${width}"/>
```



**Рисунок 61. Вид компонента выбора даты**

Для выбора валюты требуется выпадающий список доступных валют. Для реализации такого компонента необходим элемент *lookupField* :

```
<lookupField id="currency"
             datasource="cardDs"
             property="currency"
             optionsDatasource="currencyDs"
             width="70"/>
```



**Рисунок 62. Вид компонента выбора валюты**

Далее необходимо зарегистрировать экран в файле `ext-web-screens.xml` модуля `web`, чтобы его можно было открывать из меню, других окон или из программного кода.

```
<screen id="ext$InvoiceForPayment.edit"
       template="/com/haulmont/ext/web/ui/invoiceforpayment/invoiceforpayment-edit.xml"/>
```

Определим представление в файле `ext-views.xml` модуля `core`.

```
<view class="com.haulmont.ext.core.entity.InvoiceForPayment"
      name="edit"
      extends="_local">
<property name="contractor"
          view="_minimal"/>
<property name="currency"
          view="_minimal"/>
<property name="budgetItem"
          view="_minimal"/>
<property name="procs"
          view="card-edit"
          lazy="true"/>
<property name="roles"
          view="card-edit"
          lazy="true"/>
<property name="attachments"
          view="doc-edit"
          lazy="true"/>
<property name="parentCard"
          view="_minimal"/>
<property name="projects"
          view="_minimal"/>
<property name="docKind"
          view="edit"/>
<property name="versionOf"
          view="_minimal"/>
</view>
```

Далее создайте класс *контроллера* для редактирования карточки, который должен быть унаследован от класса `com.haulmont.thesis.web.ui.basic.editor.AbstractCardEditor`, содержащего общие принципы работы с карточками, привязанными к *процессам*.

В методе `setItem()` написан блок для первоначальной инициализации счета на оплату. В каждой новой карточке проставляется создатель карточки и номер карточки с помощью *нумератора*.

```
public void setItem(Entity item) {
    super.setItem(item);

    //получение сущности
    InvoiceForPayment invoiceForPayment = (InvoiceForPayment) getItem();

    initAttachments(invoiceForPayment);

    //блок для первоначальной инициализации новой карточки
    if (PersistenceHelper.isNew(invoiceForPayment)) {
        //установка создателя в карточку заявки
        invoiceForPayment.setCreator
            (UserSessionProvider.getUserSession().getUser());
        //установка замещаемого создателя в карточку заявки
        invoiceForPayment.setSubstitutedCreator
            (UserSessionProvider.getUserSession()
                .getCurrentOrSubstitutedUser());

        //установка вида документа:
```

```

LoadContext ctx = new LoadContext(DocKind.class);
ctx.setQueryString("select dk from df$DocKind dk where dk.docType = " +
                    "(select dt from df$DocType dt where dt.name = :typeName)" +
                    .addParameter("typeName", "ext$InvoiceForPayment");
ctx.setView("edit");
DataService dataService = ServiceLocator.lookup(DataService.NAME);
DocKind docKind = dataService.load(ctx);
invoiceForPayment.setDocKind(docKind);

//установка времени
invoiceForPayment.setDateTime(TimeProvider.currentTimeMillis());

//если нумератор, подключенный к данной карточке
//имеет тип "При создании"
if
(NumeratorType.ON_CREATE.equals(invoiceForPayment.getDocKind().getNumeratorType())) {
    //то получаем данный нумератор
    NumerationService ns = ServiceLocator.lookup(NumerationService.NAME);
    //получение номера из последовательности
    String num = ns.getNextNumber(invoiceForPayment);
    if (num != null)
        //с помощью метода доступа
        //проставляем номер в документ
        invoiceForPayment.setNumber(num);
}

//проставление "родительской" карточки в текущую
Card parentCard = ((Doc) item).getParentCard();
if (parentCard != null) {
    invoiceForPayment.setParentCard(parentCard);
}
}

//удаление нотификаций для данной карточки
deleteNotifications(invoiceForPayment);

//обновление папок приложений
((DocflowAppWindow) App.getInstance().getAppWindow())
    .reloadAppFolders(Collections.singletonList(invoiceForPayment));

//получение процесса, доступного для данной карточки
Proc process = invoiceForPayment.getProc();
boolean isActiveProc = false;
/* AbstractWfAccessData */
accessData = getContext().getParamValue("accessData");
if (!(accessData == null || accessData.getStartCardProcessEnabled()))
    isActiveProc = true;
else if (process != null)
    for (UUID uuid : cardProcDs.getItemIds()) {
        CardProc cpt = cardProcDs.getItem(uuid);
        if (cpt.getProc().getId().equals(process.getId()) &&
            && cpt.getActive())
            isActiveProc = true;
    }
}

//если процесс активен для данной карточки
if (!isActiveProc) {
    //делаем видимой панель с кнопками,
    //отражающими действия для данного процесса
    startProc.setVisible(true);
    CollectionDatasource<Proc, UUID> procDs = cardProcFrame
        .getDsContext().get("procDs");
}

```

```

        if (procDs.getState() == Datasource.State.VALID) {
            for (UUID uuid : procDs.getItemIds()) {
                final Proc proc = procDs.getItem(uuid);

                //создание кнопки запуска процесса
                Button button = new WebButton();
                button.setWidth("200px");
                button.setAction(new AbstractAction("start" +
                    proc.getJbpmProcessKey()) {
                    public void actionPerformed(Component component) {
                        startProcess(proc);
                    }
                });

                @Override
                public String getCaption() {
                    return proc.getName();
                }
            );
            buttonStart.add(button);
        }
    }

    if (isTabComment)
        ((TabSheet) getComponent("tabsheet")).setTab("cardCommentTab");
}

```

В методе `init()` происходит добавление слушателей источников данных. Например, для того чтобы при добавлении вложений их количество сразу отображалось в названии вкладки, нужно написать следующий слушатель:

```

public void init(Map<String, Object> params) {
    /*...*/
    /*добавление слушателя, который следит за изменением
    количества вложений в карточке
    если вложения есть, то в название вкладки добавляется число вложений
    в данной карточке*/
    attachmentsDs.addListener(new CollectionDsListenerAdapter() {
        @Override
        public void collectionChanged(CollectionDatasource ds, Operation operation) {
            for (TabSheet.Tab tab : ((TabSheet) getComponent("tabsheet")).getTabs()) {
                if (tab.getName() != null && tab.getName().equals("attachmentsTab")) {
                    if (ds.getItemIds().size() > 0)
                        tab.setCaption(getMessage("attachments") +
                            " (" + ds.getItemIds().size() + ")");
                    else
                        tab.setCaption(getMessage("attachments"));
                }
            }
        }
    }

    @Override
    public void itemChanged(Datasource ds, Entity prevItem, Entity item) {
        for (Action action : attachmentsTable.getActions()) {
            if (((Card) getItem()).getJbpmProcessId() != null
                && !"actions.Create".equals(action.getId())
                && !"actions.Copy".equals(action.getId()))
                action.setEnabled(item instanceof CardAttachment);
        }
    }
}

```

```
});  
/*...*/  
}
```

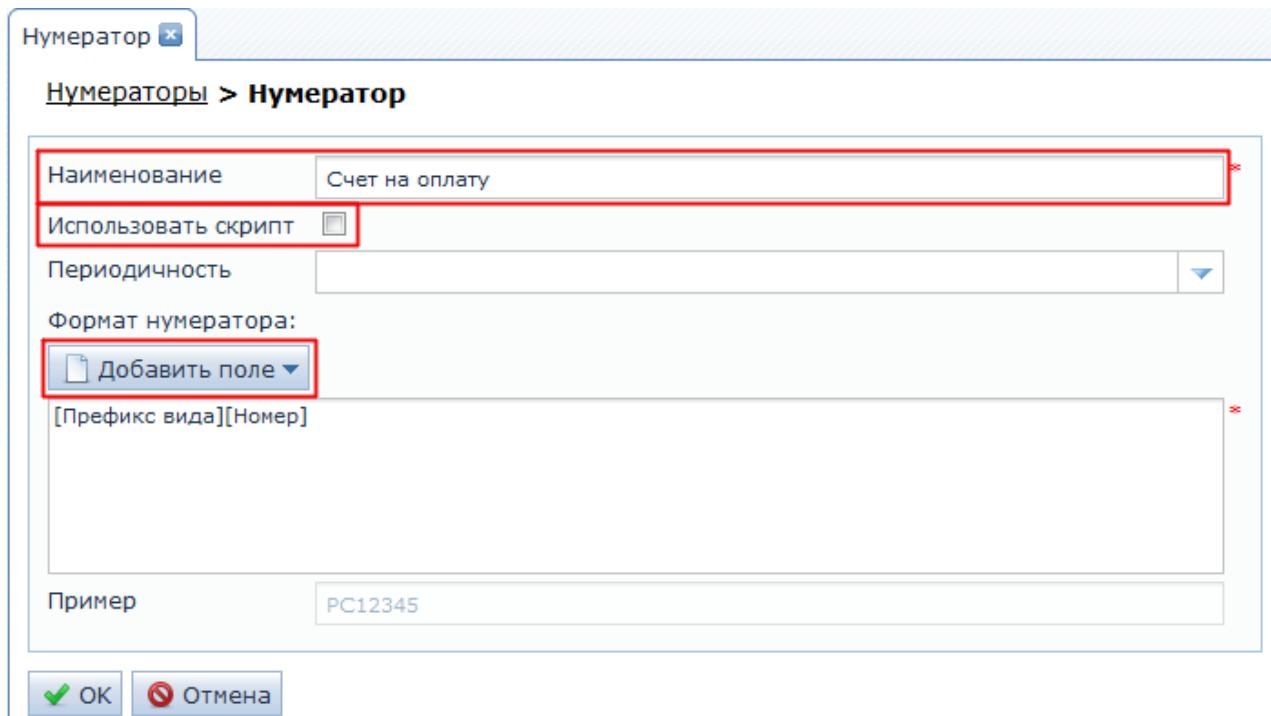
Аналогично создаются слушатели для вкладок **Проекты** и **Обсуждение**.

В методе `initcom.haulmont.thesis.web.ui.basic.editor.ActionsFrame` инициализируется панель раздела действий *процесса*, которая находится в правом верхнем углу.

Для корректного отображения названий измените *файлы локализации* `messages*.properties` в пакете `com.haulmont.ext.web.ui.invoiceforgayment` модуля `web`.

### 4.3. Создание нумератора

Для создания нумератора в системе **ТЕЗИС** перейдите к пункту меню **Администрирование** → **Нумераторы** и нажмите кнопку **Создать**.



**Рисунок 63. Окно создания нумератора**

В отобразившейся вкладке введите **Наименование** нумератора, снимите галочку **Использовать скрипт** и задайте **Формат нумератора** следующим образом: нажмите на кнопку **Добавить поле** и выберите из выпадающего списка значение **Префикс**

**вида**. Затем снова нажмите на кнопку и выберите из списка значение **Номер**.

Далее вместо [Префикс вида] напишите со-. В текстовом поле **Пример** Вы увидите, как будет выглядеть номер с созданным форматом нумератора: СО-12345.

Далее подключим созданный нумератор к карточке. Для этого перейдите к пункту меню **Документы и договоры** → **Виды документов**. Выделите из списка видов документа нужный вид карточки и нажмите кнопку **Изменить**. Перед Вами откроется окно, представленное на рисунке:

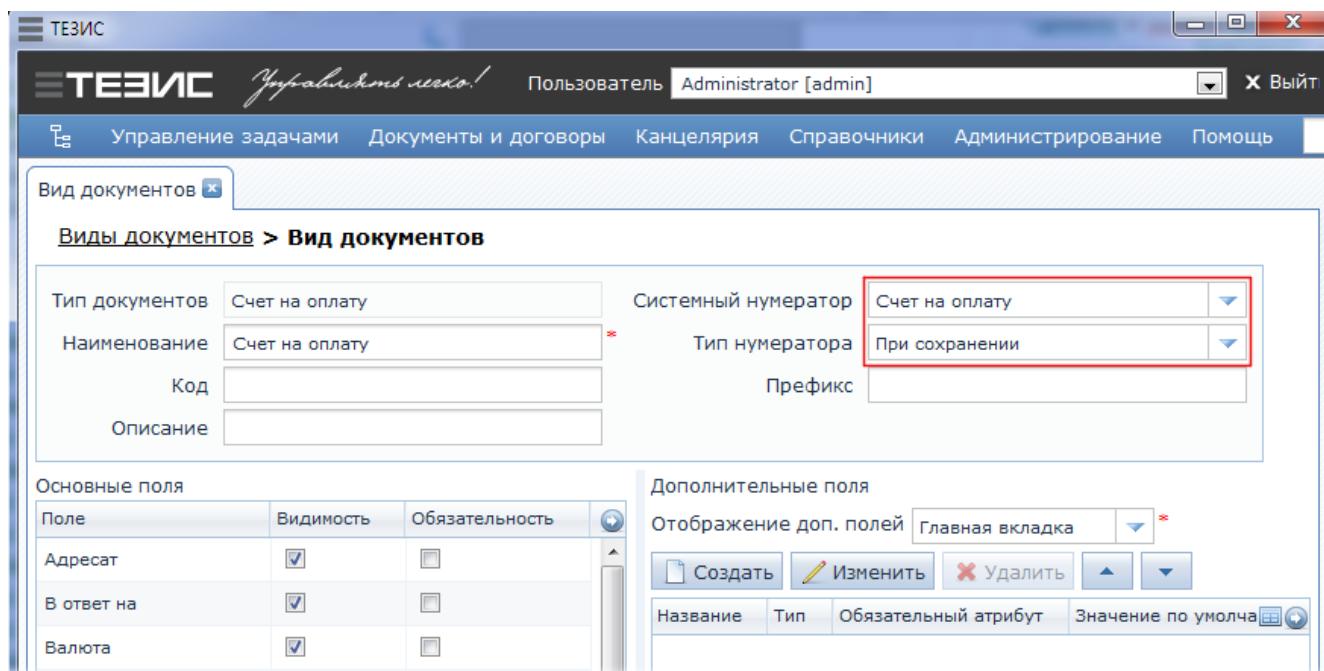


Рисунок 64. Окно редактирования вида документа Счет на оплату

В выпадающем списке **Системный нумератор** выберите созданный ранее нумератор, в качестве типа нумератора Вы можете указать **При сохранении** или **При создании**.

#### 4.4. Создание ролей

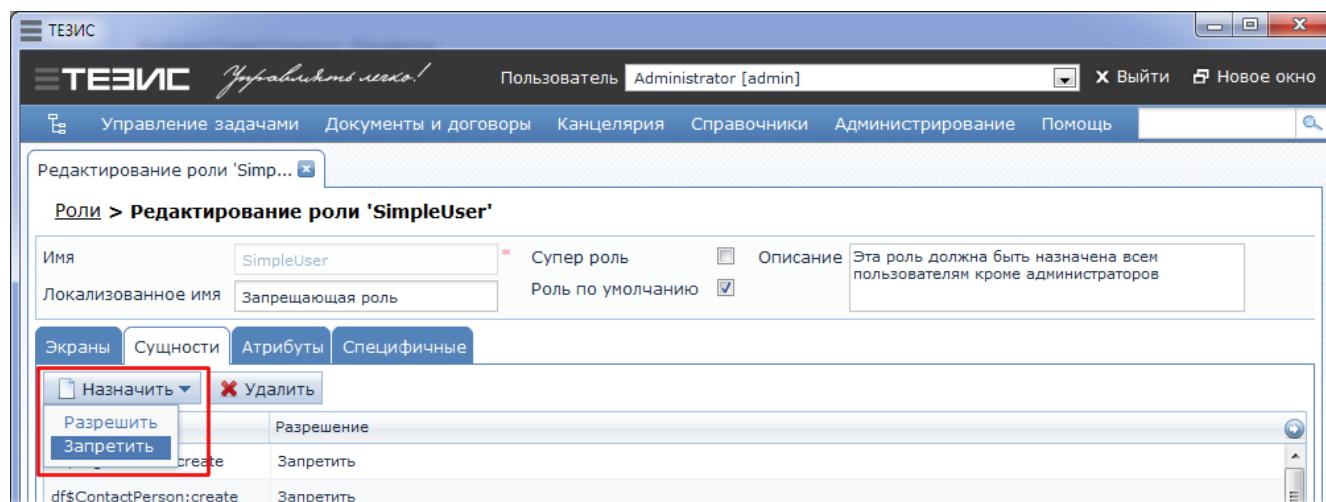
Для того чтобы запретить пользователям просматривать, создавать или изменять определенные сущности системы, существует роль *SimpleUser*. В первую очередь необходимо настроить эту роль таким образом, чтобы обладающий этой ролью пользователь не мог создавать следующие сущности системы:

- ext\$BudgetItem
- ext\$InvoiceForPayment

- `ext$Post` (с информацией о том, как создать эту сущность, Вы можете ознакомиться в главе Быстрый старт)
- `ext$Employee` (с информацией о том, как создать эту сущность, Вы можете ознакомиться в главе Быстрый старт)

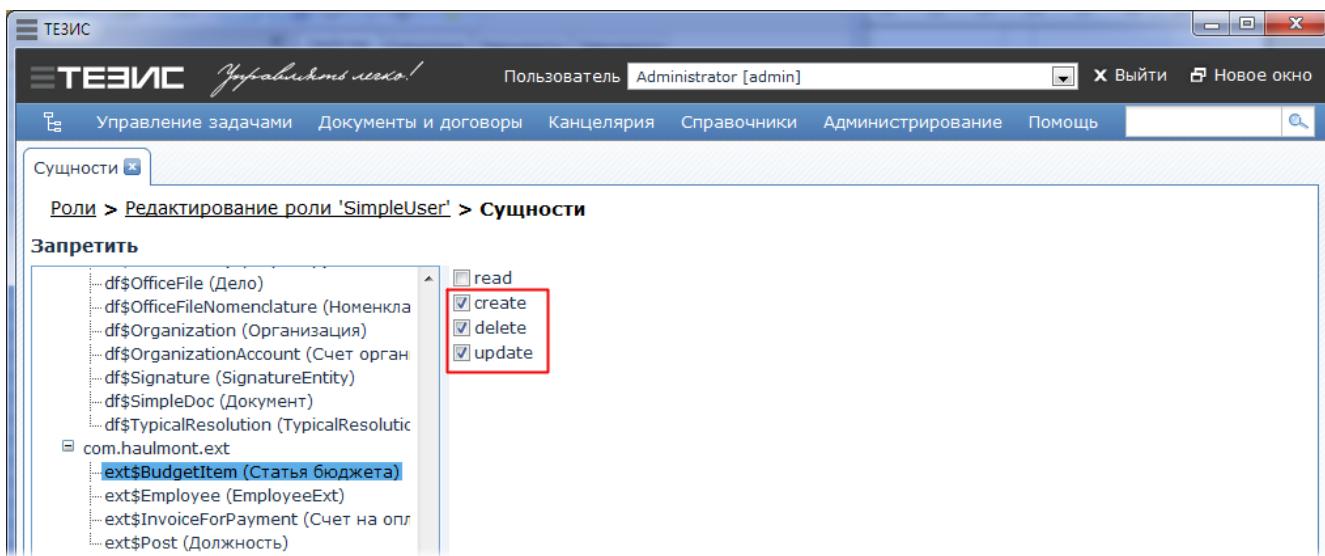
Далее покажем, как запретить создание, удаление и изменение сущности на примере `ext$BudgetItem`.

Перейдите в режим редактирования роли *SimpleUser*, далее – на вкладку **Сущности**. Нажмите на кнопку **Назначить** и в выпадающем списке выберите пункт **Запретить**.



**Рисунок 65. Окно редактирования роли SimpleUser**

В отобразившемся окне в списке слева найдите и выделите сущность `ext$BudgetItem` и отметьте галочки **create**, **delete**, **update**.



**Рисунок 66. Окно редактирования роли SimpleUser**

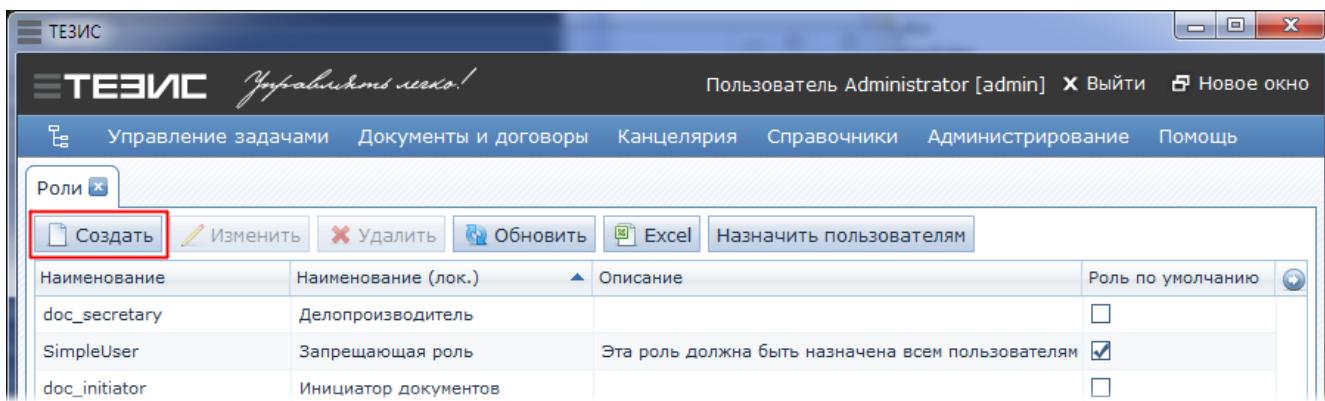
После этого нажмите на кнопку **Выбрать**.

Перечислим роли, которые необходимо создать в системе.

- *Инициатор счетов на оплату (invoice\_initiator)* может создавать статьи бюджета и счета на оплату, а также запускать процесс оплаты счета.
- *Согласующий счета на оплату (invoice\_endorsement)* может либо согласовать счет на оплату, либо отправить его на доработку.
- *Утверждающий счета на оплату (invoice\_approver)* может утвердить счет на оплату или отправить его на доработку.
- *Проверяющий счета на оплату (invoice\_checking)* проверяет правильность счета и может отправить его на оплату либо вернуть на доработку.
- *Оплачивающий счета (invoice\_paying)* может оплатить счет или отправить его на доработку.

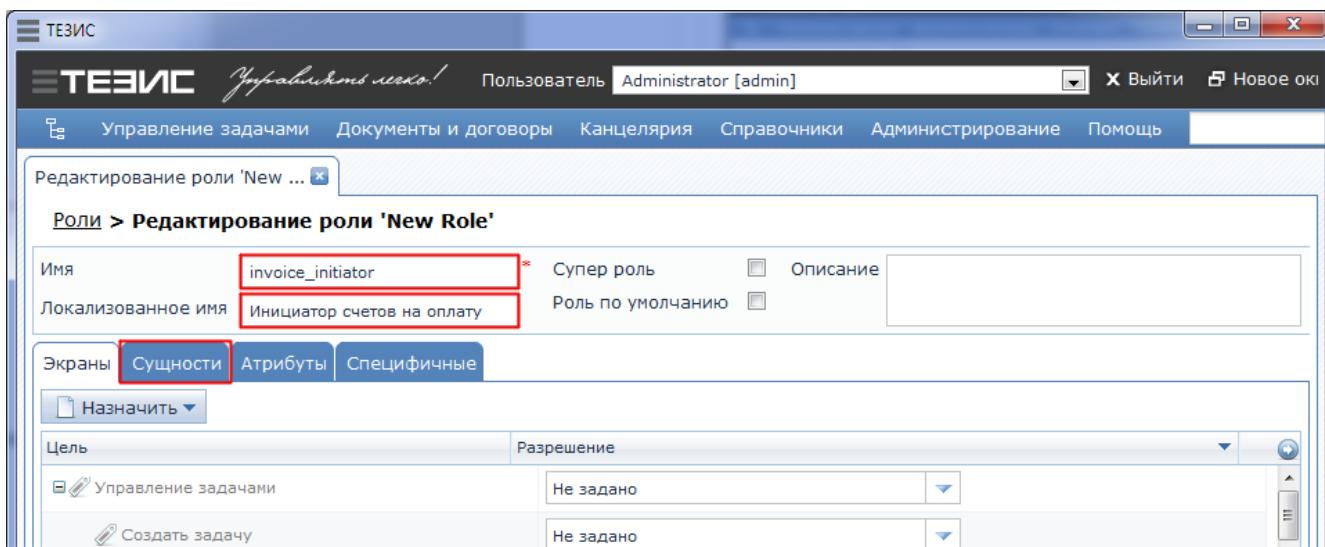
Рассмотрим более подробно, как создать роль *Инициатор счетов на оплату (invoice\_initiator)*.

Зайдите в систему под ролью **Администратор**. Выберите пункт меню **Администрирование -> Роли**. Перед Вами откроется окно, представленное на рисунке:



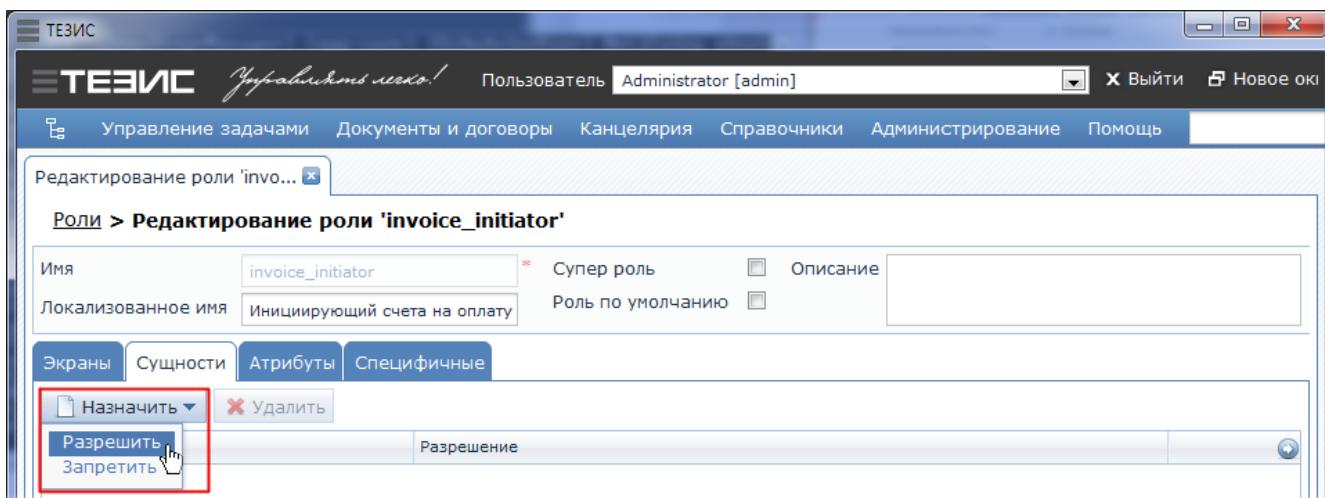
**Рисунок 67. Окно управления ролями**

Нажмите на кнопку **Создать**.



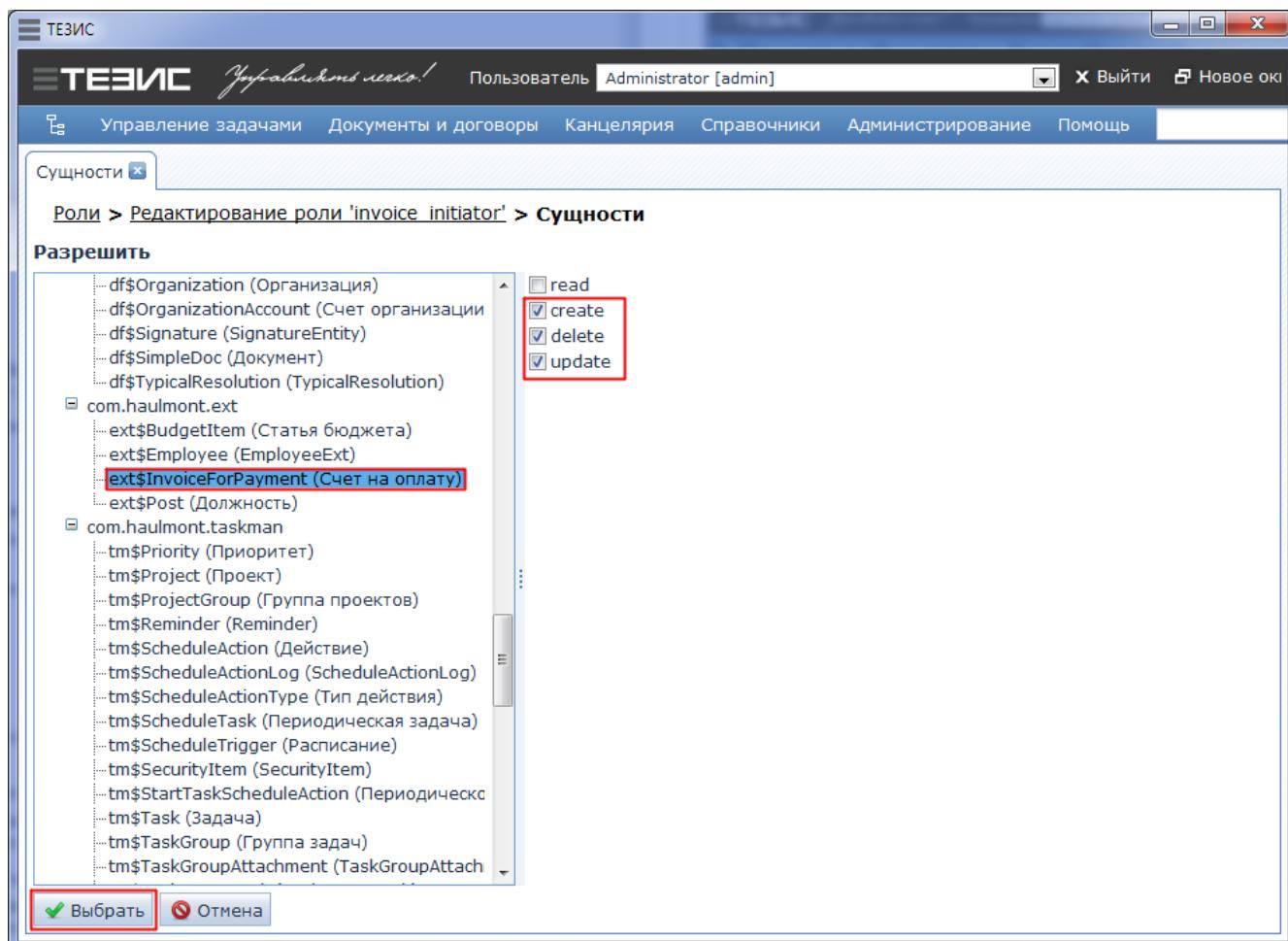
**Рисунок 68. Окно создания роли invoice\_initiator**

Далее в отобразившемся окне заполните поля **Имя** и **Локализованное имя**. Затем перейдите на вкладку **Сущности**. Нажмите на кнопку **Назначить** и в выпадающем списке выберите пункт **Разрешить**.



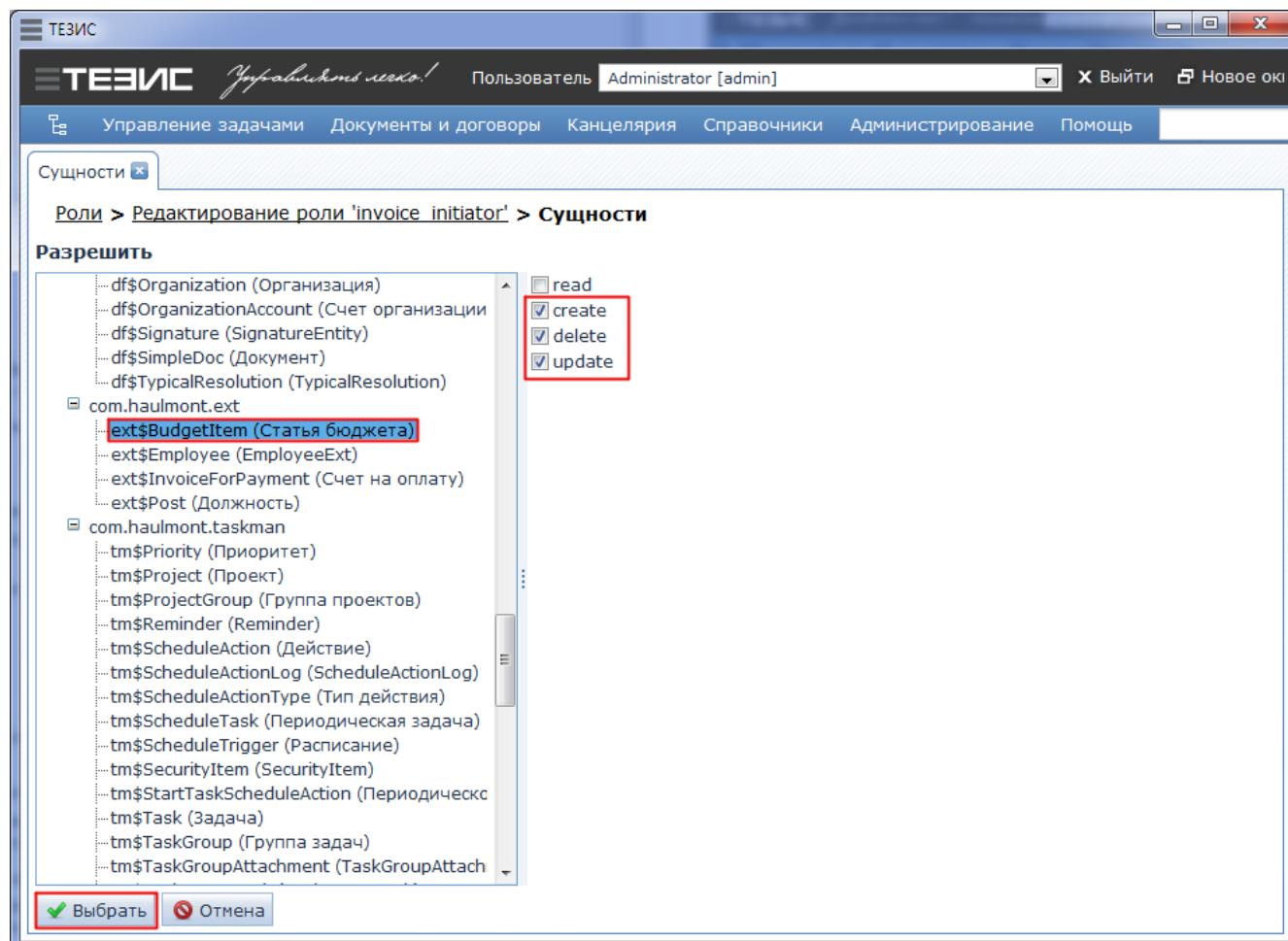
**Рисунок 69. Окно создания роли invoice\_initiator**

В отобразившемся окне в списке слева найдите и выделите сущность `ext$InvoiceForPayment` и отметьте галочки `create` , `delete` , `update` . После этого нажмите на кнопку **Выбрать**.



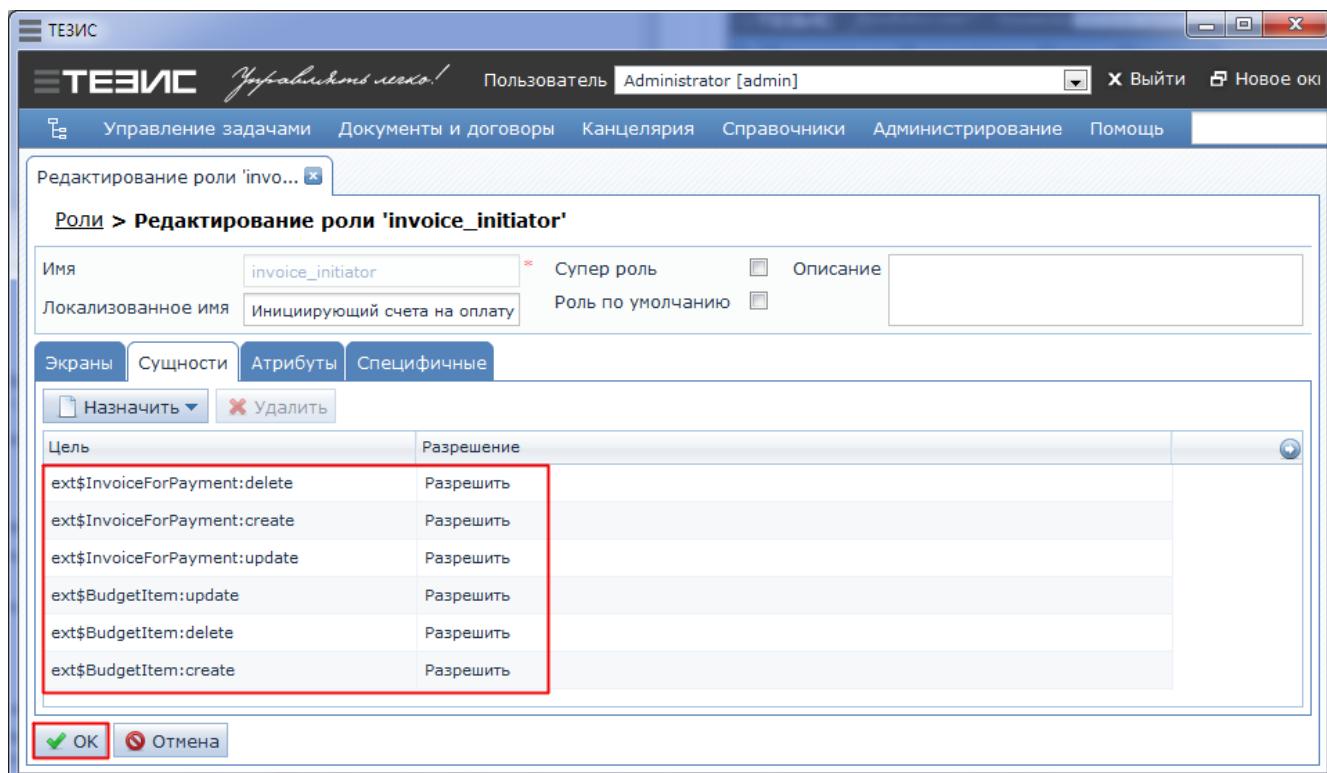
**Рисунок 70. Окно создания роли invoice\_initiator**

Затем выделите сущность ext\$BudgetItem и повторите действия, описанные выше.



**Рисунок 71. Окно создания роли invoice\_initiator**

Убедитесь, что на вкладке сущности отображается список разрешающих действий и нажмите на кнопку **OK**.

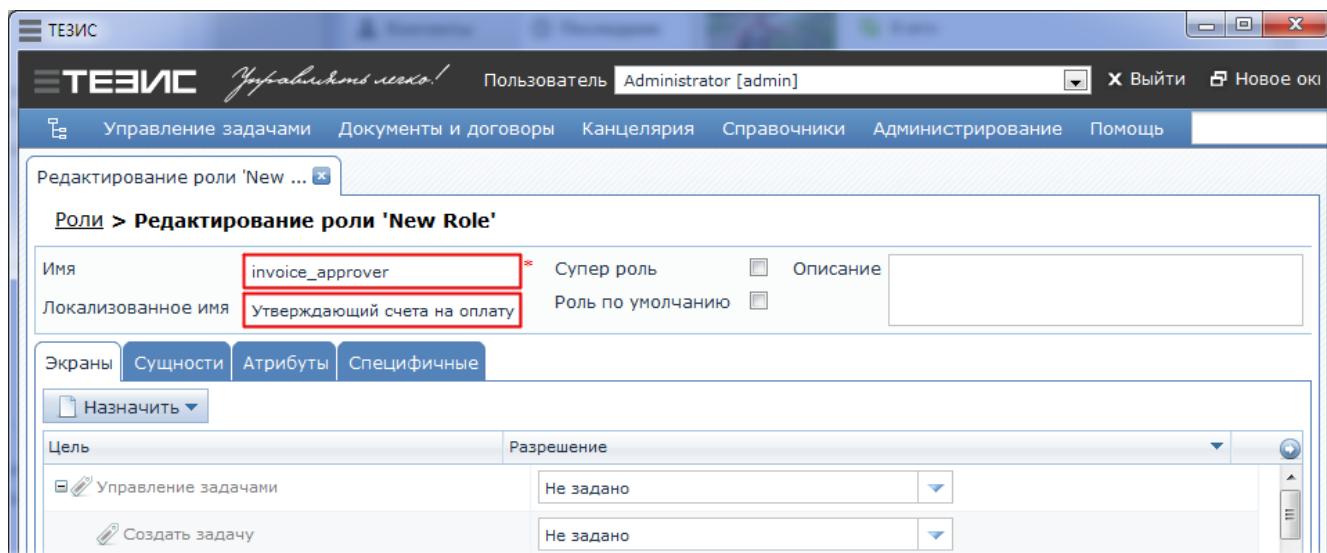


**Рисунок 72. Окно создания роли invoice\_initiator**

Оставшиеся четыре роли создаются проще. Создадим роль **Утверждающий счета на оплату (invoice\_approver)**.

В экране просмотра ролей пользователей нажмите на кнопку **Создать**.

Далее в отобразившемся окне заполните поля **Имя** и **Локализованное имя** и нажмите на кнопку **OK**.



**Рисунок 73. Окно создания роли invoice\_approver**

Аналогично создайте следующие роли:

- Согласующий счета на оплату (*invoice\_endorsement*)
- Оплачивающий счета (*invoice\_paying*)
- Проверяющий счета на оплату (*invoice\_paying*)

#### Примечание

Так как пользователь *Администратор* должен обладать всеми правами, задайте пользователю *Администратор* созданные Вами роли.

## 4.5. Редактирование групп доступа

В группах доступа задаются ограничения для большого числа пользователей. Более подробно Вы сможете ознакомиться с управлением группами доступа в разделе 2.2 *Руководства администратора* системы ТЕЗИС .

Изменим группы доступа *Ограниченный доступ*, *Руководитель департамента* и *Руководитель подразделения*.

Для этого перейдите к пункту меню **Администрирование** → **Группы доступа**. В отобразившемся окне на панели слева выделите мышкой группу доступа *Ограниченный доступ*, а на панели справа перейдите на вкладку **Ограничения** и

нажмите на кнопку **Создать**.

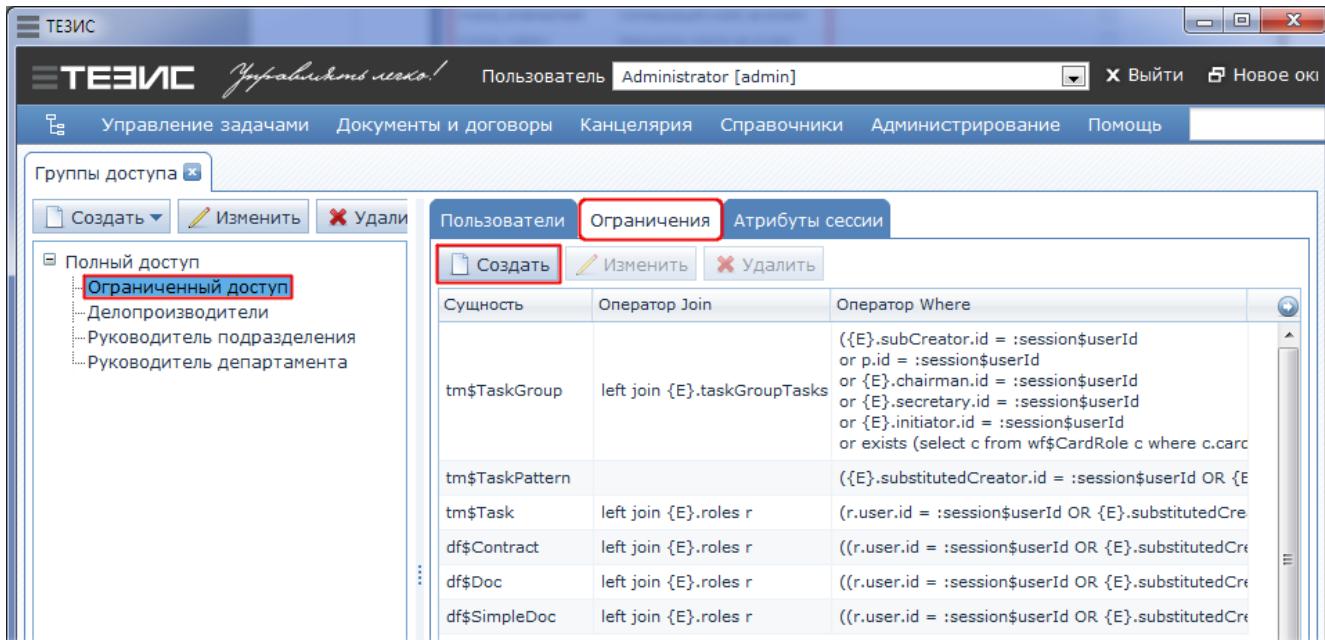


Рисунок 74. Окно управления группами доступа

Далее заполните поля **Имя сущности**, **Оператор Join**, **Оператор Where** и нажмите на кнопку **OK**.

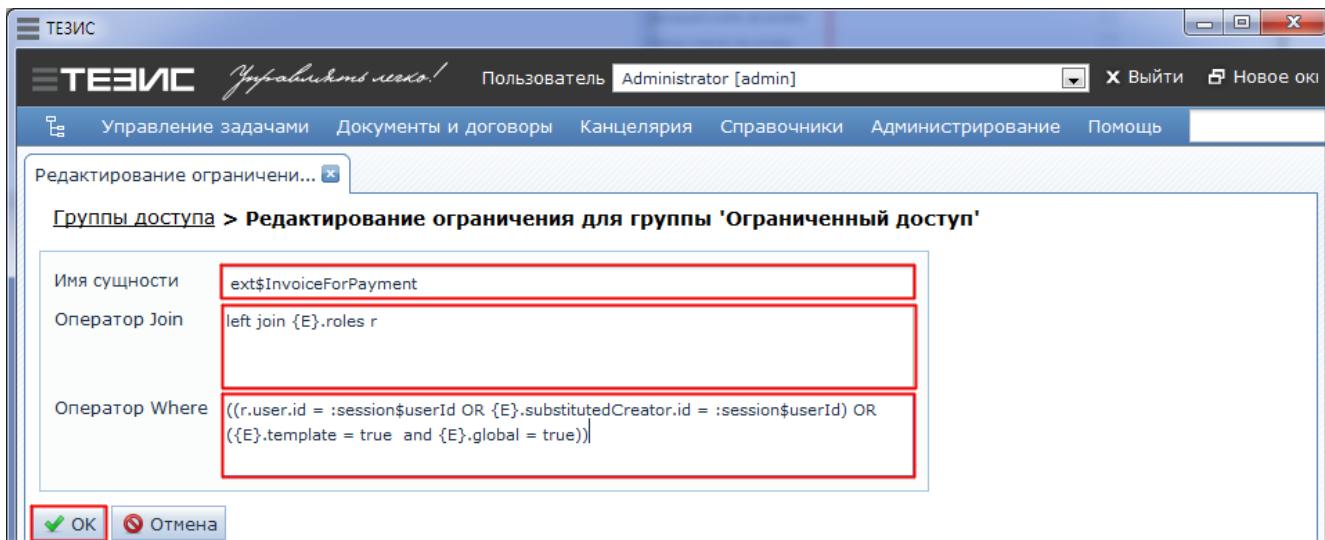


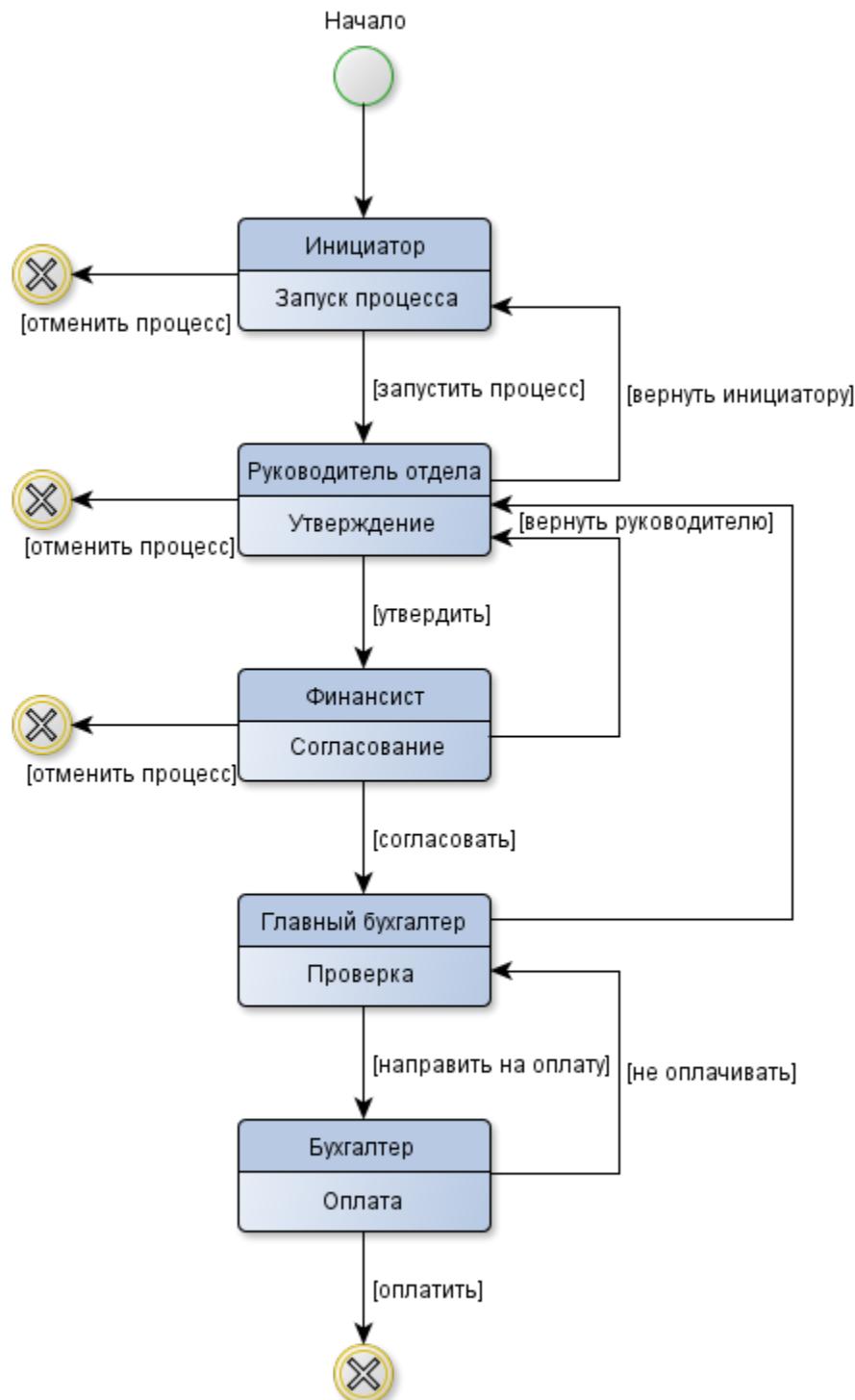
Рисунок 75. Редактирование ограничения для группы Ограниченный доступ

По аналогии создайте ограничения для групп доступа *Руководитель подразделения* и *Руководитель департамента*. Значения Вы можете взять из таблицы, расположенной ниже.

Название поля	Значение
<b>Ограничения для группы Ограниченный доступ</b>	
Имя сущности	ext\$InvoiceForPayment
Оператор Join	left join {E}.roles r
Оператор Where	((r.user.id = :session\$userId OR {E}.substitutedCreator.id = :session\$userId) OR ({E}.template = true and {E}.global = true))
<b>Ограничения для группы Руководитель департамента</b>	
Имя сущности	ext\$InvoiceForPayment
Оператор Join	left join {E}.roles r left join {E}.substitutedCreator u
Оператор Where	(r.user.id = :session\$userId OR u.id = :session\$userId OR ({E}.template = true and {E}.global = true) OR exists(select em from df\$Employee em where (em.user.id = r.user.id or em.user.id = u.id) and em.department.id in (:session\$departmentIds)))
<b>Ограничения для группы Руководитель подразделения</b>	
Имя сущности	ext\$InvoiceForPayment
specialOperator Join	left join {E}.roles r left join {E}.substitutedCreator u
Оператор Where	(r.user.id = :session\$userId OR u.id = :session\$userId OR ({E}.template = true and {E}.global = true) OR exists(select em from df\$Employee em where (em.user.id = r.user.id or em.user.id = u.id) and em.department.id in (select eu.department.id from df\$Employee eu where eu.user.id = :session\$userId)))

## 4.6. Создание процесса оплаты счета

Создадим процесс оплаты счета. Графически данный процесс можно представить следующим образом:



**Рисунок 76. Процесс оплаты счета**

Для того чтобы войти в конструктор бизнес-процессов, в главном меню системы

**ТЕЗИС** выберите пункт меню **Администрирование**, далее – подпункт **Дизайн процессов**.

Для дальнейших инструкций обратитесь к Главе 2.11 *Руководства пользователя системы ТЕЗИС*.

Создайте в окне дизайнера процесс "Оплата счета":

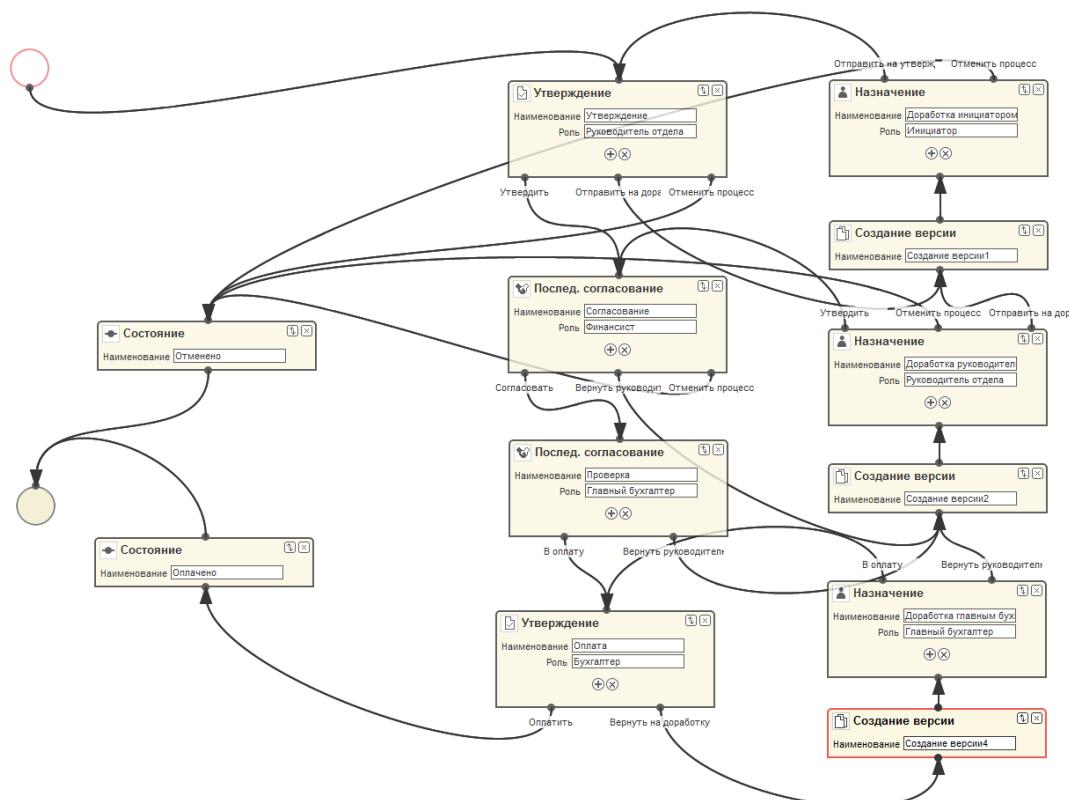


Рисунок 77. Процесс "Оплата счета" в дизайнере процессов

В каждом модуле необходимо задать вид перехода.

Если переход успешный, кнопка, соответствующая данному переходу, будет иметь рамочку зеленого цвета. Если переход неуспешный, то кнопка будет иметь рамочку красного цвета.

### Внимание

Порядок выходов в модулях имеет значение! Слева обязательно должен быть успешный переход.

**Процесс**

Наименование

**Модуль**

Наименование

Описание

Цифровая подпись

▼ Формы

---

Переход	Удалить
<input type="button" value="Утвердить"/>	

**Вид перехода**

---

**Форма**

Вложения

Необходим комментарий

Цифровая подпись

---

Переход	Удалить
<input type="button" value="Отправить на доработку"/>	

**Вид перехода**

---

**Форма**

Вложения

Необходим комментарий

Цифровая подпись

---

Переход	Удалить
<input type="button" value="Отменить процесс"/>	

**Вид перехода**

---

**Форма**

Вложения

Необходим комментарий

Цифровая подпись



**Действия**

Документы

### Рисунок 78. Панель редактирования переходов

После создания дизайна процесса можно выгрузить шаблон матрицы оповещений, который представляет из себя файл .xls. На вкладке **ACTIONS** по умолчанию определены три типа оповещений для документов и договоров. Переопределим их для счетов на оплату.

**SIMPLE (оповещения серого цвета):**

```

import com.haulmont.cuba.core.SecurityProvider
import com.haulmont.cuba.core.global.ConfigProvider
import com.haulmont.cuba.core.global.GlobalConfig
import com.haulmont.workflow.core.entity.Card
import com.haulmont.cuba.core.global.MessageUtils
import org.apache.commons.lang.StringUtils
import com.haulmont.thesis.core.notification.NotificationUtils
import com.haulmont.cuba.security.entity.User
import com.haulmont.cuba.core.global.MessageProvider
import com.haulmont.ext.core.entity.InvoiceForPayment

User u = user
String link = makeLink(card)
User cu = SecurityProvider.currentUserSession().getCurrentOrSubstitutedUser()

String dateFormat = MessageProvider.getMessage(MessageUtils.getMessagePack(),
    &quot;dateFormat&quot;)

String userName = (u.lastName != null ? u.lastName : &quot;&quot;)+&quot; &quot;+
    (u.firstName != null ? u.firstName : &quot;&quot;)+&quot; &quot;+
    (u.middleName != null ? &quot; &quot; + u.middleName : &quot;&quot;)

String cuName = (cu.lastName != null ? cu.lastName : &quot;&quot;)+&quot; &quot;+
    (cu.firstName != null ? cu.firstName : &quot;&quot;)+&quot; &quot;+
    (cu.middleName != null ? &quot; &quot; + cu.middleName : &quot;&quot;)

String lang = u.language ? u.language :
    SecurityProvider.currentUserSession().getLocale().getLanguage()
String email = StringUtils.isBlank(u.email)?&quot;&quot;:u.email;

if (lang == &apos;ru&apos;) {
    if (card instanceof InvoiceForPayment) {
        InvoiceForPayment c = card
        subject = &quot;Уведомление по счету на оплату №
${(StringUtils.isNotBlank(c.number))}?&quot;&quot;
            :c.number} от ${c.date.format(dateFormat)}&quot;
        body = &quot;&quot;&quot;
        &lt;html&gt;&lt;body&gt;
        ${userName}! &lt;br /&gt;
        &lt;b&gt;Счет на оплату:&lt;/b&gt; №
${(StringUtils.isNotBlank(c.number))}?&quot;&quot;:c.number}
            от ${c.date.format(dateFormat)} перешел в состояние
${c.getLocState()}&lt;br&gt;
        &lt;b&gt;Текущий процесс:&lt;/b&gt;
        ${(c.proc!=null)?c.proc.name:&quot;нет&quot;}&lt;br /&gt;
        &lt;b&gt;Список участников процесса:&lt;/b&gt;&lt;br /&gt;&lt;br /&gt;
        ${NotificationUtils.getListOfParticipantsOfProcess(c, lang)}
            Для открытия карточки счета на оплату перейдите по
                &lt;a href=&quot;${link}&quot;&gt;этой ссылке&lt;/a&gt;.
                &lt;br /&gt;
        Сообщение отправлено автоматически. Пожалуйста, не отвечайте на него.
        &lt;/body&gt;&lt;/html&gt;
        &quot;&quot;&quot;
    }
}
```

```

    }
}

else{
    if (card instanceof InvoiceForPayment){
        InvoiceForPayment c = card
        subject = "Notification by invoice for payment #"
            ${StringUtils.isBlank(c.number)}?"":c.number}
            from ${c.date.format(dateFormat)}";
        Body = "Invoice for payment:</b> #"
${(StringUtils.isNotBlank(c.number))?"":c.number}
        from ${c.date.format(dateFormat)} change state ${c.getLocState()}<br>;
        <b>Current process:</b>
        ${c.proc!=null}?c.proc.name:"no&br />
<b>List of process participants:</b>&lt;br /&gt;&lt;br /&gt;&lt;br /&gt; $"
        {NotificationUtils.getListOfParticipantsOfProcess(c, lang)}
        To open the invoice click <a href=${link}>${link}</a> to card</a>.
        &lt;br />
        This message was sent automatically. Please do not reply.
        &lt;/body&gt;&lt;/html&gt;
        """
    }
}

String getInstanceName(Card card) {
    return ''ext$InvoiceForPayment';
}

String makeLink(Card c) {
    GlobalConfig conf = ConfigProvider.getConfig(GlobalConfig.class)
    return ""${conf.webAppUrl}/open?"+
        ""screen=${getInstanceName(c)}.edit&"+
        ""item=${getInstanceName(c)}-${c.id}&"+
        ""params=item:${getInstanceName(c)}-${c.id}"";
}

```

## ACTION (оповещения зеленого цвета)

```

import com.haulmont.cuba.core.SecurityProvider
import com.haulmont.cuba.core.global.ConfigProvider
import com.haulmont.cuba.core.global.MessageProvider
import com.haulmont.cuba.core.global.GlobalConfig
import com.haulmont.workflow.core.entity.Card
import com.haulmont.cuba.core.global.MessageUtils
import org.apache.commons.lang.StringUtils
import com.haulmont.thesis.core.notification.NotificationUtils
import com.haulmont.cuba.security.entity.User
import com.haulmont.ext.core.entity.InvoiceForPayment

User u = user
String link = makeLink(card)
User cu = SecurityProvider.currentUserSession().getCurrentOrSubstitutedUser()

String dateFormat = MessageProvider.getMessage(MessageUtils.getMessagePack(),
    "dateFormat");

String userName = (u.lastName != null ? u.lastName : "")+" "+
    (u.firstName != null ? u.firstName : "")+" "+
    (u.middleName != null ? " " + u.middleName : "");
String cuName = (cu.lastName != null ? cu.lastName : "")+" "+
    (cu.firstName != null ? cu.firstName : "")+" "+
    (cu.middleName != null ? " " + cu.middleName : "");

```

```

String lang = u.language ? u.language : SecurityProvider.currentUserSession()
    .getLocale().getLanguage()
String email = StringUtils.isNotBlank(u.email)?":&u.email

if (lang == &apos;ru&apos;) {
    if (card instanceof InvoiceForPayment) {
        InvoiceForPayment c = card
        subject = "Требуется ваше участие по счету на оплату №
            ${StringUtils.isNotBlank(c.number)}:&c.number} от
            ${c.date.format(dateFormat)}";
        body = "<html><body>
            ${userName}!
            <br />
        Требуется ваше участие на оплату № ${c.number} от
            ${c.date.format(dateFormat)}<br />
            <br />
            Текущее состояние:<b> ${c.getLocState()}</b><br />
            <br />
            Текущий процесс:<b> ${c.proc!=null?c.proc.name}</b>
            <br />
            <br />
            Список участников процесса:<b> ${NotificationUtils.getListOfParticipantsOfProcess(c, lang)}
            Для открытия карточки счета на оплату перейдите по
                <a href="${link}">этой ссылке</a>.
            <br />
            Сообщение отправлено автоматически. Пожалуйста, не отвечайте на него.
            <br />
            <br />
        }>
    }
}
else {
    if (card instanceof InvoiceForPayment) {
        InvoiceForPayment c = card
        subject = "Your participation in Invoice for payment # ${StringUtils.isNotBlank(c.number)}:&c.number} from
            ${c.date.format(dateFormat)} is required";
        Body = "<html><body>
            ${userName}!
            <br />
        Requires your participation through the invoice for payment #
            ${StringUtils.isNotBlank(c.number)}:&c.number} from
            ${c.date.format(dateFormat)}<br />
            <br />
            Current state:<b> ${c.getLocState()}</b><br />
            <br />
            Current process:<b> ${c.proc!=null?c.proc.name}</b>
            <br />
            <br />
            List of process participants:<b> ${NotificationUtils.getListOfParticipantsOfProcess(c, lang)}
            To open the invoice for payment click
                <a href="${link}">go to card</a>.
            <br />
            This message was sent automatically. Please do not reply.
            <br />
            <br />
        }>
    }
}

String getInstanceName(Card card) {
    return &apos;ext$InvoiceForPayment&apos;;
}

String makeLink(Card c) {
    GlobalConfig conf = ConfigProvider.getConfig(GlobalConfig.class)
    return "${conf.webAppUrl}/open?" +
        "screen=${getInstanceName(c)}.edit&quot; +

```

```

    &quot;item=${getInstanceName(c)}-${c.id}&amp;&quot; +
    &quot;params=item:${getInstanceName(c)}-${c.id}&quot;
}

```

## WARNING (оповещения красного цвета)

```

import com.haulmont.cuba.core.SecurityProvider
import com.haulmont.cuba.core.global.ConfigProvider
import com.haulmont.cuba.core.global.GlobalConfig
import com.haulmont.workflow.core.entity.Card
import com.haulmont.cuba.core.global.MessageUtils
import org.apache.commons.lang.StringUtils
import com.haulmont.thesis.core.notification.NotificationUtils
import com.haulmont.cuba.security.entity.User
import com.haulmont.ext.core.entity.InvoiceForPayment
import com.haulmont.cuba.core.global.MessageProvider

User u = user
String link = makeLink(card)
User cu = SecurityProvider.currentUserSession().getCurrentOrSubstitutedUser()

String dateFormat = MessageProvider.getMessage(MessageUtils.getMessagePack(),
    &quot;dateFormat&quot;)

String userName = (u.lastName != null ? u.lastName : &quot;&quot;)+&quot; &quot;+
    (u.firstName != null ? u.firstName : &quot;&quot;)+&quot; &quot;+
    (u.middleName != null ? &quot; &quot; + u.middleName : &quot;&quot;)
String cuName = (cu.lastName != null ? cu.lastName : &quot;&quot;)+&quot; &quot;+
    (cu.firstName != null ? cu.firstName : &quot;&quot;)+&quot; &quot;+
    (cu.middleName != null ? &quot; &quot; + cu.middleName : &quot;&quot;)
String lang = u.language ? u.language : SecurityProvider.currentUserSession()
    .getLocale().getLanguage()
String email = StringUtils.isBlank(u.email)?&quot;:&quot;:u.email;

if (lang == &apos;ru&apos;) {
    if (card instanceof InvoiceForPayment){
        InvoiceForPayment c = card
        subject = &quot;Обратите внимание на счет на оплату №
            ${StringUtils.isBlank(c.number)}?&quot;&quot;:c.number} от
            ${c.date.format(dateFormat)}&quot;
        body = &quot;&quot;&quot;
        &lt;html&gt;&lt;body&gt;
        ${userName}! &lt;br /&gt;
        Обратите внимание на счет на оплату №
        ${StringUtils.isBlank(c.number)}?&quot;&quot;:c.number} от
        ${c.date.format(dateFormat)}&lt;br /&gt;
        &lt;b&gt;Текущее состояние:&lt;/b&gt; ${c.getLocState()}&lt;br /&gt;
        &lt;b&gt;Текущий процесс:&lt;/b&gt;
        ${c.proc!=null?c.proc.name:&quot;нет&quot;}&lt;br /&gt;
        &lt;b&gt;Список участников процесса:&lt;/b&gt; &lt;br /&gt;&lt;br /&gt;
        ${NotificationUtils.getListOfParticipantsOfProcess(c, lang)}
        Для открытия карточки счета на оплату перейдите
        по &lt;a href=&quot;${link}&quot;&gt;этой ссылке&lt;/a&gt;.
        &lt;br/&gt;
        Сообщение отправлено автоматически. Пожалуйста, не отвечайте на него.
        &lt;/body&gt;&lt;/html&gt;
        &quot;&quot;&quot;
    }
}
else{
    if (card instanceof InvoiceForPayment){
        InvoiceForPayment c = card
        subject = &quot;Pay your attention to Invoice for payment #
            ${StringUtils.isBlank(c.number)}?&quot;&quot;:c.number} from

```

```

    ${c.date.format(dateFormat)}";
    body = "":";
    &lt;html&gt;&lt;body&gt;
    ${userName}! &lt;br /&gt;
Pay your attention to invoice for payment #
${(StringUtils.isBlank(c.number))?":":c.number} from
${c.date.format(dateFormat)}&lt;br /&gt;
    &lt;b&gt;Current state:&lt;/b&gt; ${c.getLocState()}&lt;br /&gt;
&lt;b&gt;Current process:&lt;/b&gt;
    ${(c.proc!=null)?c.proc.name:"no"}&lt;br /&gt;
&lt;b&gt;List of process participants:&lt;/b&gt;&lt;br /&gt;&lt;br /&gt;
${NotificationUtils.getListOfParticipantsOfProcess(c, lang)}
To open the card click &lt;a href="${link}"&gt;go to card&lt;/a&gt;.
&lt;br /&gt;
This message was sent automatically. Please do not reply.
&lt;/body&gt;&lt;/html&gt;
""";
}
}

String getInstanceName(Card card) {
    return &apos;ext$InvoiceForPayment&apos;;
}

String makeLink(Card c) {
    GlobalConfig conf = ConfigProvider.getConfig(GlobalConfig.class)
    return "${conf.webAppUrl}/open?&quot; +
        &quot;screen=${getInstanceName(c)}.edit&amp;quot; +
        &quot;item=${getInstanceName(c)}-${c.id}&amp;quot; +
        &quot;params=item:${getInstanceName(c)}-${c.id}&quot;;
}

```

После компиляции и развертывания процесса необходимо указать, что с этим процессом могут работать только карточки счетов на оплату. Для этого в меню **Администрирование -> Процессы** выберите необходимый процесс "Оплата счета" и нажмите кнопку **Изменить**.

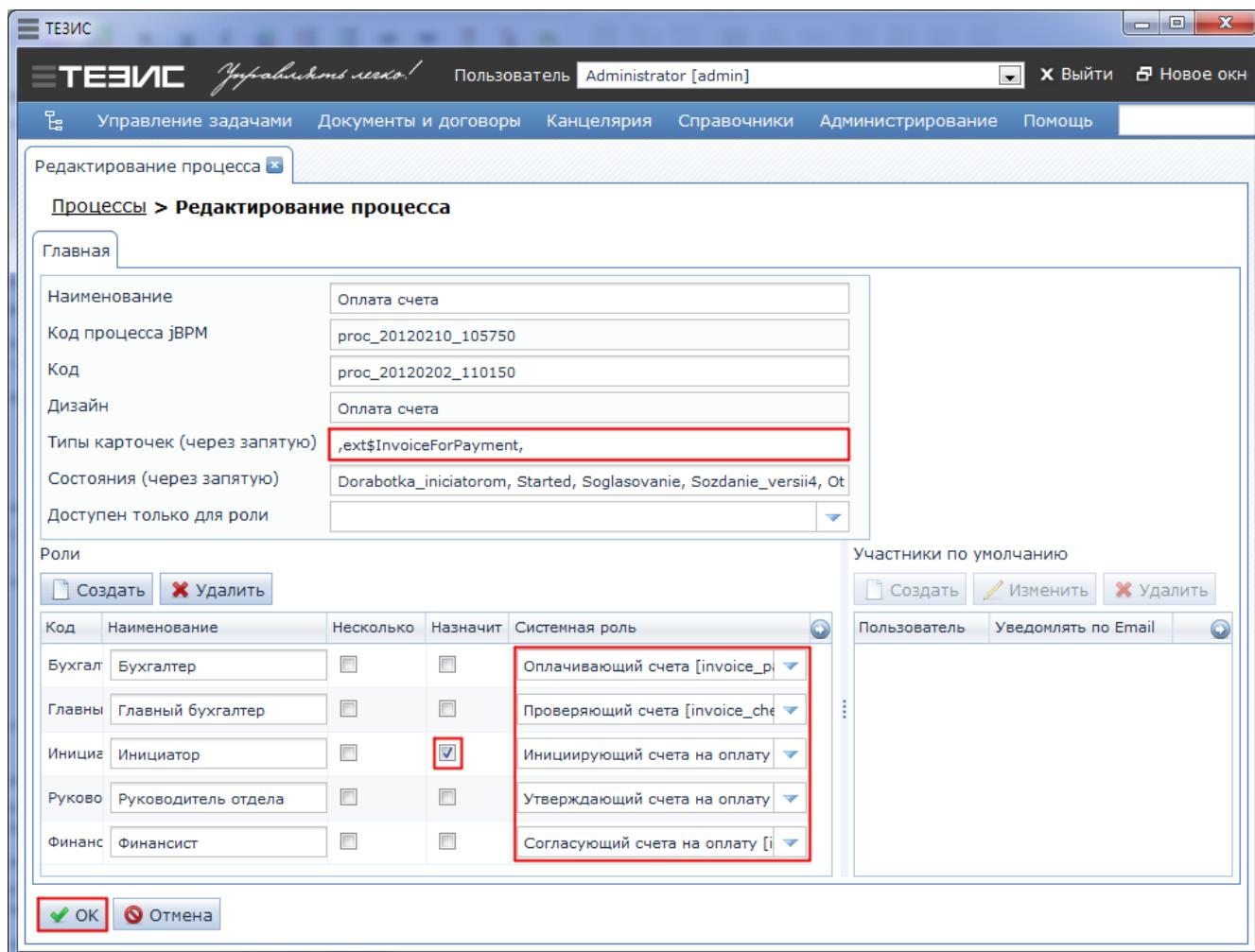


Рисунок 79. Окно редактирования процесса

В отобразившемся экране в поле **Типы карточек** введите `,ext$InvoiceForPayment,`. Также Вам потребуется задать системные роли для процесса "Оплата счета", которые были созданы в разделе 4.4. Соответствие ролей процесса и системных ролей отражено в таблице:

Роль процесса	Системная роль
Бухгалтер	Оплачивающий счета (invoice_paying)
Главный бухгалтер	Проверяющий счета на оплату (invoice_checking)
Инициатор	Инициатор счетов на оплату (invoice_initiator)
Руководитель отдела	Утверждающий счета на оплату (invoice_approver)
Финансист	Согласующий счета на оплату (invoice_endorsement)

В колонке **Назначить на создателя** поставьте галочку роли *Инициатор*.

Дизайн процесса и матрица оповещений находятся в проекте в пакете com.haulmont.ext.web модуля web.

## 4.7. Создание папок приложения

Панель папок предназначена для быстрого доступа пользователя к часто используемой информации. В системе **ТЕЗИС** поддерживается два вида папок:

- *Папки приложения*
- *Папки поиска*

Для описания папок приложения используются экземпляры сущности com.haulmont.cuba.core.entity.AppFolder. Атрибуты сущности:

- *filterComponentId* – путь к компоненту-фильтру, включая имя экрана. Может содержать только имя экрана, в этом случае фильтр не используется.
- *filterXml* – фильтр в формате XML.
- *visibilityScript* – путь к скрипту Groovy, в котором определяется, нужно ли отображать данную папку. Скрипт должен вернуть значение типа boolean.
- *quantityScript* – путь к скрипту Groovy, который будет выполнен для подсчета количества записей в данной папке. Скрипт должен вернуть значение типа Number.

Заметим, что в *quantityScript* подпапки **Документы** → **Новые** подсчитывается количество всех карточек, унаследованных от сущности Doc.

```
package com.haulmont.thesis.core.appfolders

import com.haulmont.cuba.core.*

Transaction tx = Locator.createTransaction()
try {
    EntityManager em = PersistenceProvider.getEntityManager()
    Query q = em.createQuery("select count(c.id) from df\\$Doc c where
        c.substitutedCreator.id = ?1 and c.state is null and c.template = false
        and c.versionOf is null");
    q.setParameter(1, SecurityProvider.currentOrSubstitutedUserId())
    def result = q.getSingleResult();
    tx.commit();
    return result
} finally {
    tx.end();
}
```

Поэтому нам придется написать новый скрипт, определяющий количество документов и договоров.

```

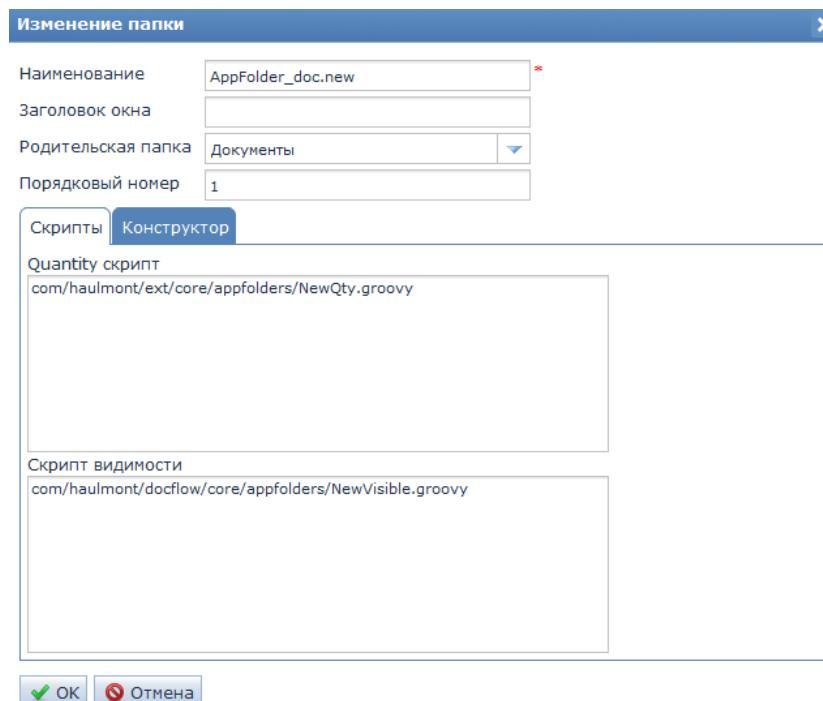
package com.haulmont.ext.core.appfolders

import com.haulmont.cuba.core.*

Transaction tx = Locator.createTransaction()
try {
    EntityManager em = PersistenceProvider.getEntityManager()
    Query q = em.createQuery("""select count(c.id) from df\$/Doc c where
        c.substitutedCreator.id = ?1 and c.state is null
        and c.template = false and c.versionOf is null
        and c.docKind.docType.name <> 'ext\$InvoiceForPayment'""")
    q.setParameter(1, SecurityProvider.currentOrSubstitutedUserId())
    def result = q.getSingleResult();
    tx.commit();
    return result
} finally {
    tx.end();
}
    
```

В этом скрипте при запросе к базе данных мы исключаем документы вида "Счет на оплату".

После написания скрипта необходимо указать путь к нему в окне изменения папки.



**Рисунок 80. Окно редактирования папки AppFolder\_doc.new**

Теперь количество новых карточек вида "Документы" и "Договоры" подсчитывается верно. Но при открытии этой папки карточки вида "Счет на оплату" все равно попадают

в список карточек. Связано это с тем, что при открытии папки вызывается экран `df$Doc.browse`, в источнике данных которого выгружаются все экземпляры сущностей, дочерних к сущности `Doc`.

Переопределим этот экран. Для этого создайте XML-дескриптор `extdoc-browse.xml` и укажите, что он унаследован от дескриптора `doc-browse.xml` путем указания в корневом элементе атрибута `extends`, содержащего ссылку на базовый дескриптор. Далее нужно переопределить источник данных `docsDs`:

```
<!window extends="/com/haulmont/thesis/web/ui/doc/doc-browse.xml">
<dsContext>
  <groupDatasource id="docsDs"
    class="com.haulmont.thesis.core.entity.Doc"
    view="browse">
    <datasourceClass>com.haulmont.thesis.web.ui.DocDatasource</datasourceClass>
    <query>
      <![CDATA[select distinct d from df$Doc d
      where d.id < :param$exclItem
      and d.template = false
      and d.versionOf is null
      order by d.dateTime]]>
    <filter>
      <and>
        <c>d.docOfficeDocKind = :param$docOfficeDocKind</c>
    </and>
    <![CDATA[d.docKind.docType.name < ; ext$InvoiceForPayment ; ]]>
    </query>
    </groupDatasource>
  </dsContext>
</window>
```

В результате переопределения экрана в подпапке **Документы -> Новые** находятся только карточки документов и договоров.

Создадим папки приложения для счетов на оплату. Папки, связанные с процессом, можно создать через интерфейс системы. Подробное описание, как это сделать, будет представлено ниже. Папку **Новые** создать через интерфейс корректно не получится, поэтому создадим для нашего проекта `MBean`, в котором будут инициализироваться папки приложения. В дальнейшем в этот бин можно будет добавлять дополнительные методы для настройки проекта, например, для журналирования сообщений.

Создадим интерфейс с именем, заканчивающимся на `MBean` и класс, реализующий этот интерфейс в пакете `com.haulmont.ext.core.app` модуля `core`.

Интерфейс `ExtensionDeployerMBean`:

```
package com.haulmont.ext.core.app;
```

```
public interface ExtensionDeployerMBean {
    String NAME = "ext_ExtensionDeployer";

    String createAppFolders();
}
```

Класс ExtensionDeployer наследуется от com.haulmont.thesis.core.app.AbstractDeployer, который содержит в себе общие методы для создания папок, и реализует интерфейс ExtensionDeployerMBean.

```
package com.haulmont.ext.core.app;

import com.haulmont.cuba.core.EntityManager;
import com.haulmont.cuba.core.Persistence;
import com.haulmont.cuba.core.PersistenceProvider;
import com.haulmont.cuba.core.Transaction;
import com.haulmont.cuba.core.entity.Folder;
import com.haulmont.cuba.core.global.MessageProvider;
import com.haulmont.ext.core.entity.InvoiceForPayment;
import com.haulmont.thesis.core.app.AbstractDeployer;
import org.apache.commons.lang.exception.ExceptionUtils;

import javax.annotation.ManagedBean;
import javax.inject.Inject;
import java.util.Arrays;
import java.util.List;

@ManagedBean(ExtensionDeployerMBean.NAME)
public class ExtensionDeployer extends AbstractDeployer
    implements ExtensionDeployerMBean{

    @Inject
    private Persistence persistence;

    /*Метод для создания папок вида:
     Счета на оплату
     -Новые(0)*/
    public String createAppFolders() {
        try {
            login();
            List<String> foldersNames = Arrays
                .asList("AppFolder_invoiceForPayment.new",
                        "AppFolder_invoiceForPayment.invoicesForPayment");
            /*удаляем все папки приложения с именами
             из списка выше при их пересоздании*/
            deleteAppFolders(foldersNames);
            Transaction tx = persistence.createTransaction();
            try {
                EntityManager em = PersistenceProvider.getEntityManager();

                //создание корневой папки "Счета на оплату"

                Folder invoicesForPaymentFolder = createAppFolder(
                    "AppFolder_invoiceForPayment.invoicesForPayment",
                    null,
                    null,
                    null,
                    null,
                    null,
                    null,
                    null,
                    null);

                "com/haulmont/ext/core/appfolders/InvoicesForPaymentVisible.groovy";
```

```

        5,
        null
    );
    em.persist(invoicesForPaymentFolder);

    //создание папки "Новые";
    em.persist(createAppFolder(
        //название фильтра:
        &quot;AppFolder_invoiceForPayment.new";,
        // [экран].id фильтра:
        &quot;[ext$InvoiceForPayment.browse].genericFilter";,
        //xml фильтра:
        getNewFolderXml(),
        //скрипт видимости:

"com/haulmont/ext/core/appfolders/NewInvoicesVisible.groovy";,
        //скрипт на количество новых карточек:
        &quot;com/haulmont/ext/core/appfolders/NewInvoicesQty.groovy";,
        1,//порядковый номер
        invoicesForPaymentFolder//родительская папка
    ));

    tx.commit();
    return &quot;App folders successfully created";
} finally {
    tx.end();
}
} catch (Exception e) {
    return ExceptionUtils.getStackTrace(e);
} finally {
    logout();
}
}

//xml фильтр для папки "Новые";
private String getNewFolderXml() {
    return &quot;<?xml version=&quot;1.0&quot;
encoding=&quot;UTF-8&quot;?>\n&quot; +
    &quot;\n&quot; +
    &quot;<filter>\n&quot; +
    &quot;  &lt;and>\n&quot; +
    &quot;    &lt;c name=&quot;new" locCaption=&quot;&quot; +
        getFolderCaption(&quot;invoiceForPayment.new") +
        &quot;    unary=&quot;true&quot; hidden=&quot;true&quot;
type=&quot;CUSTOM"; entityAlias=&quot;t"; &quot; +
    &quot;      join=&quot;*&quot;&gt;i.substitutedCreator.id = :session$userId
and i.state is null &quot; +
    &quot;      and i.template = false and i.versionOf is null\n&quot; +
    &quot;      &lt;param
name=&quot;component$genericFilter.new821667"&gt;true&lt;/param&gt;\n&quot; +
    &quot;      &lt;/c&gt;\n&quot; +
    &quot;      &lt;/and&gt;\n&quot; +
    &quot;      &lt;/filter&gt;&quot;;
}

private String getFolderCaption(String folder) {
    return MessageProvider.getMessage(this.getClass(), folder);
}
}
}

```

После того как создали MBean , его необходимо зарегистрировать его в файле ext-spring.xml :

```

<bean id="ext_ExtensionExporter"
      class="org.springframework.jmx.export.MBeanExporter"
      lazy-init="false">
    <property name="beans">
      <map>
        <entry key="${cuba.webContextName}:service=ExtensionDeployer"
               value-ref="ext_ExtensionDeployer"/>
      </map>
    </property>
    <property name="assembler">
      <bean
        class="org.springframework.jmx.export.assembler.InterfaceBasedMBeanInfoAssembler">
        <property name="interfaceMappings">
          <map>
            <entry key="ext_ExtensionDeployer"
                   value="com.haulmont.ext.core.app.ExtensionDeployerMBean"/>
          </map>
        </property>
      </bean>
    </property>
  </bean>

```

Убедимся, что созданный MBean появился в системе. Для этого перейдите к пункту меню **Администрирование** → **Консоль JMX**.

В отобразившемся окне нажмите на знак плюса напротив домена **app-core** и найдите в раскрывшемся списке строку **app-core:service=ExtensionDeployer**. Далее нажмите на кнопку **Просмотреть MBean**.

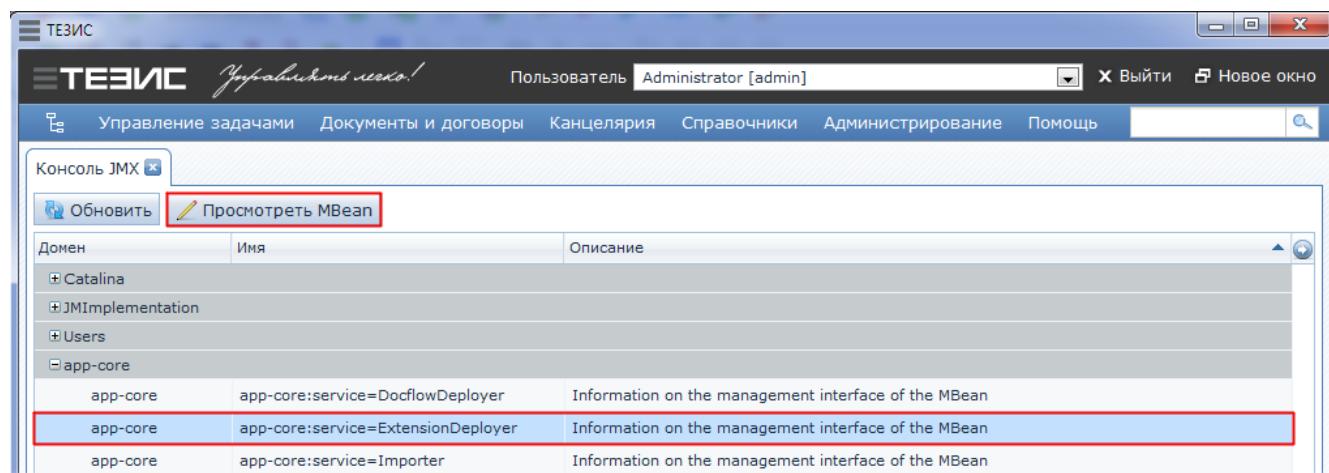
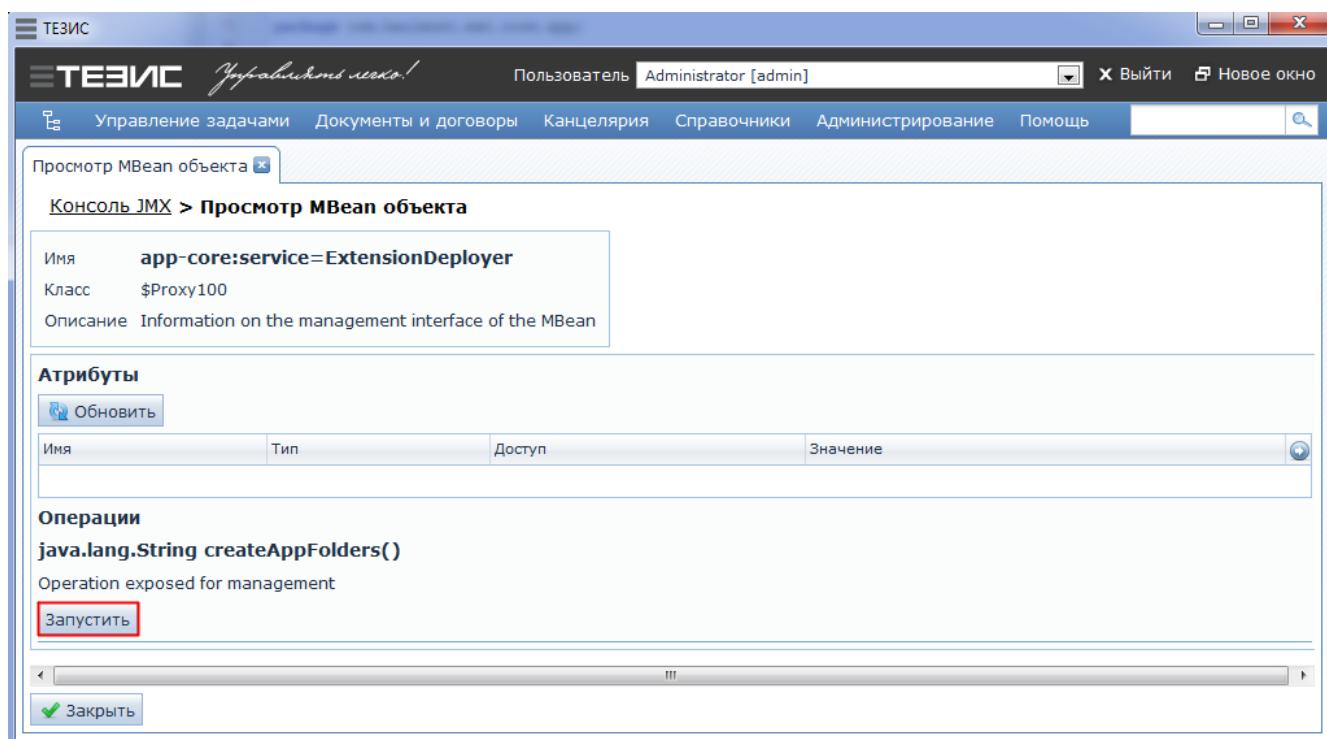


Рисунок 81. Консоль JMX

В отобразившемся окне Вы можете увидеть, что ExtensionDeployer в секции **Операции** имеет метод `createAppFolders()`, созданный нами ранее.



**Рисунок 82. Окно просмотра MBean объекта ExtensionDeployer**

Нажмите на кнопку **Запустить** метода `createAppFolders()`. После этого откроется окно с сообщением о результатах выполнения операции. Если операция была выполнена неуспешно, рекомендуется просмотреть журналы сообщений Tomcat.

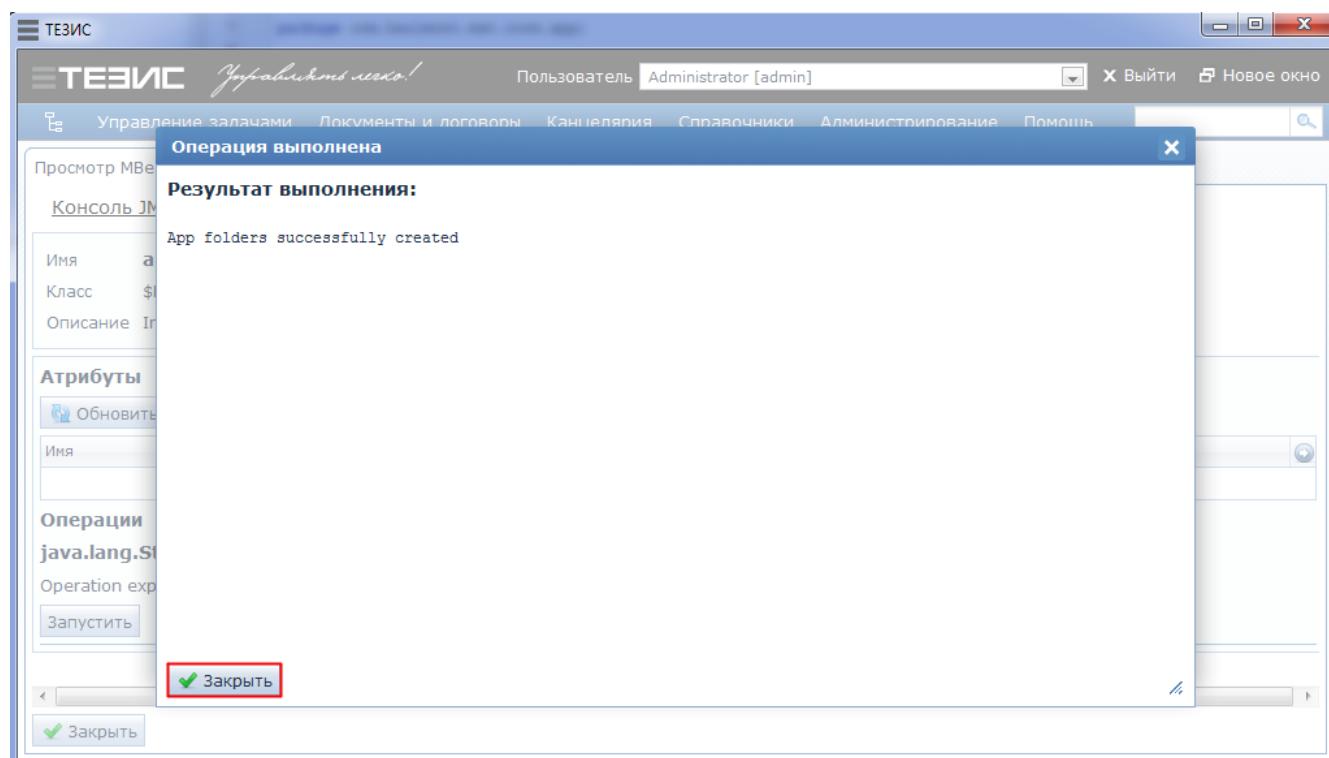


Рисунок 83. Окно с сообщением о результатах выполнения операции

После успешного завершения операции папки приложения примут вид, представленный на рисунке ниже.

#### Папки приложения

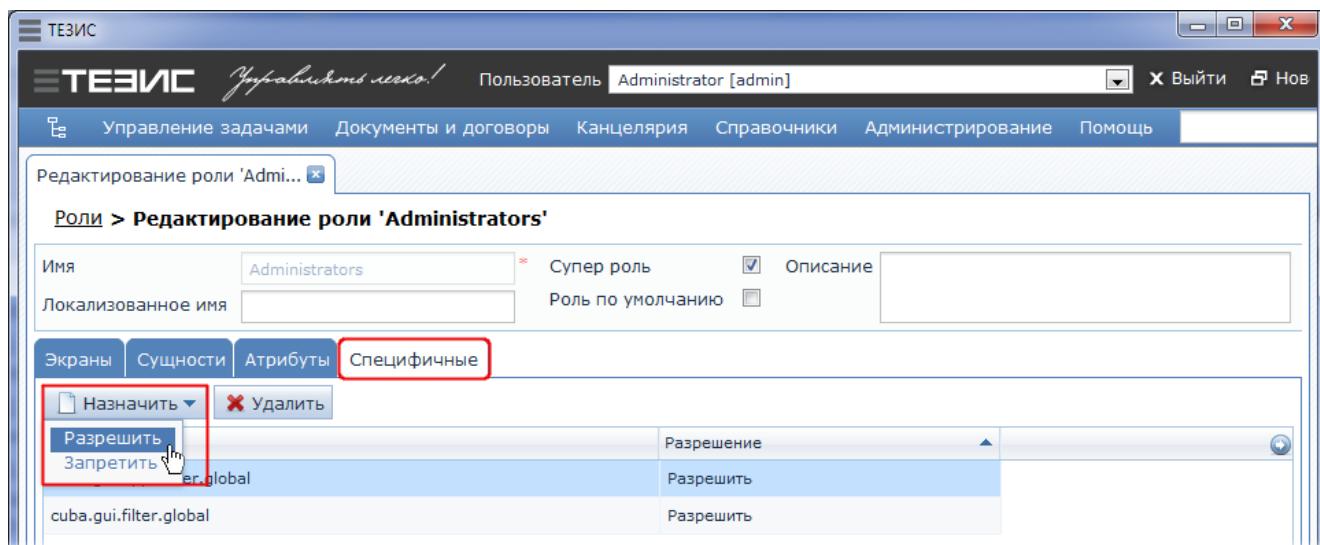
- Задачи
  - Новые (0)
  - Возврат от исполнителя (0)
  - Назначенные (0)
  - В работе (0)
  - Для контроля (0)
  - Наблюдаемые (0)
- Документы
  - Новые (1)
  - Согласование (0)
  - Доработка (0)
  - Утверждение (0)
  - Ознакомление (0)
- Канцелярия
  - Регистрация (0)
  - Резолюция (0)
  - Обработка резолюции (0)
- Счета на оплату
  - Новые (0)

Рисунок 84. Папки приложения

Далее создадим папки приложения для процесса "Оплата счета" с помощью интерфейса.

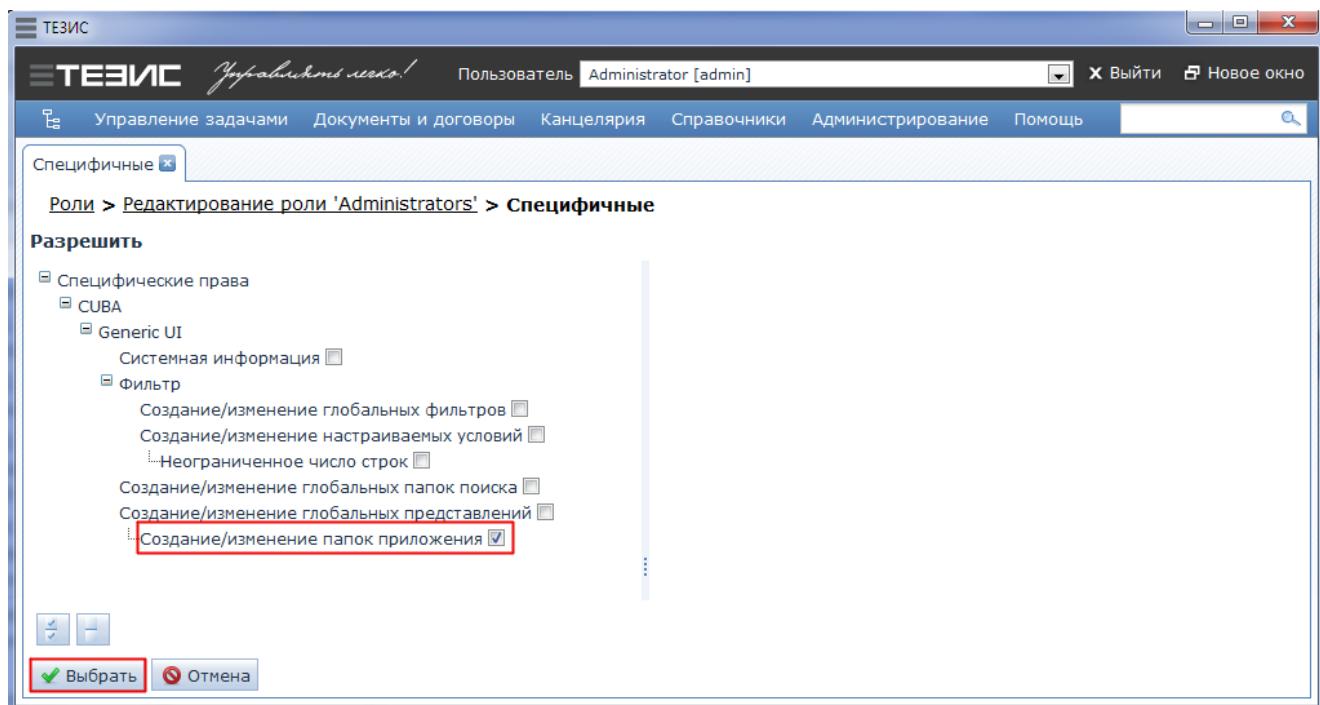
#### 4.7.1. Создание папок приложения, привязанных к процессу

Для начала необходимо настроить права таким образом, чтобы пользователь с ролью **Администратор** имел возможность создавать папки приложения. Для этого зайдите в систему **ТЕЗИС** под пользователем **Администратор**. Далее в пункте меню выберите **Администрирование -> Роли**. Выделите роль **Администратор** и нажмите на кнопку **Изменить**. Далее перейдите на вкладку **Специфичные** и нажмите на кнопку **Назначить -> Разрешить**.



**Рисунок 85. Окно редактирования роли Administrator**

В отобразившемся окне установите галочку **Создание/изменение папок приложения** и нажмите на кнопку **Выбрать**.



**Рисунок 86. Окно редактирования роли Administrator**

Теперь пользователь с ролью **Администратор** сможет создавать новые папки приложения , изменять и удалять существующие папки приложения.

Для процесса "Согласование счета" создадим следующие папки приложения:

1. **Утверждение**. В эту папку у пользователя *Руководитель отдела* будут попадать счета на оплату, отправленные на утверждение, а также счета, вернувшиеся к нему на доработку.
2. **Доработка** содержит счета на оплату, вернувшиеся от *Руководителя отдела* на доработку *Инициатору*.
3. **Согласование**. В этой папке отображаются счета на оплату, попавшие *Финансисту* на согласование.
4. **Проверка**. В эту папку попадают счета, отправленные на проверку *Главному бухгалтеру*, а также вернувшиеся на доработку от *Бухгалтера*.
5. **К оплате** содержит счета на оплату, попавшие на оплату пользователю *Бухгалтер*.

Рассмотрим процесс создания папки приложения на примере папки **Доработка счета**.

Для того чтобы создать папку, нажмите правой кнопкой мыши на папке **Счета на оплату**, далее в отобразившемся контекстном меню выберите пункт **Создать**.

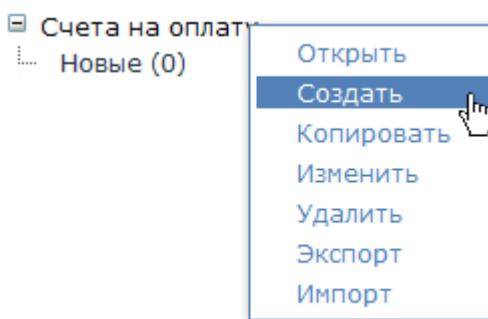


Рисунок 87. Контекстное меню для папок приложения

Перед Вами откроется окно, представленное на рисунке ниже.

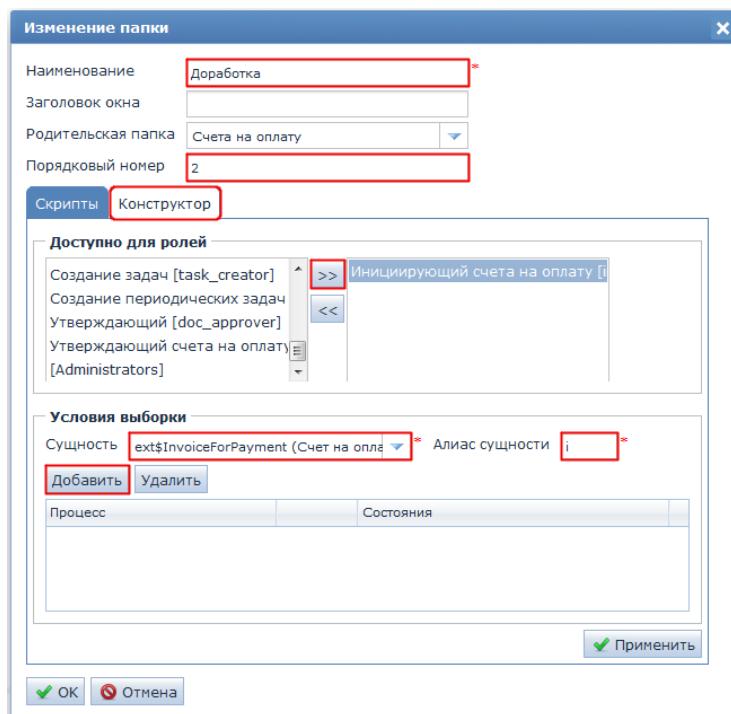


Рисунок 88. Окно редактирования папки Доработка

В поле ввода **Наименование** введите название папки – **Доработка**. В поле ввода **Порядковый номер** задается порядковый номер данной папки в списке папок. Далее перейдите на вкладку **Конструктор**. В панели **Доступно для ролей** из списка слева выберите роль *Иницирующий счета на оплату* путем нажатия на кнопку **>>**. В панели **Условия выборки** выберите сущность *ext\$InvoiceForPayment* (*Счет на оплату*) из списка предложенных сущностей.

### Внимание

В качестве алиаса сущности укажите то же самое название, которые Вы указывали в *дескрипторе экрана invoiceforpayment-browse.xml*.

Далее укажем, при каком условии в папку **Доработка** будут поступать карточки счетов на оплату. Для этого нажмите на кнопку **Добавить**.

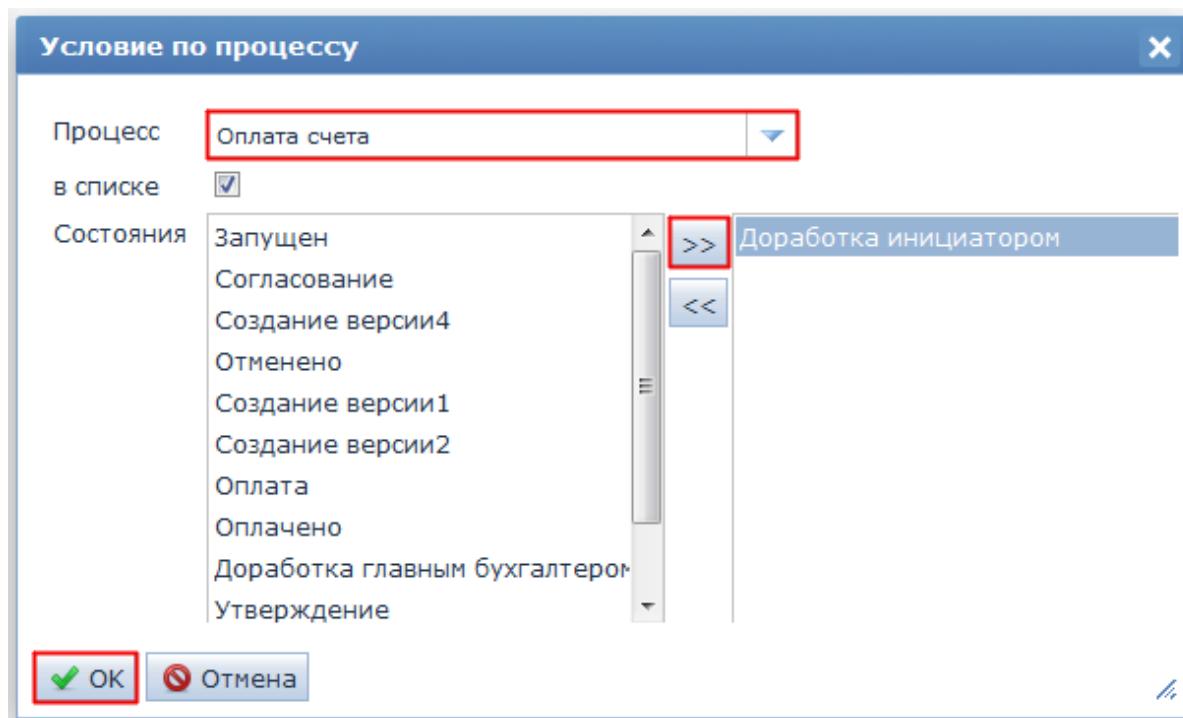
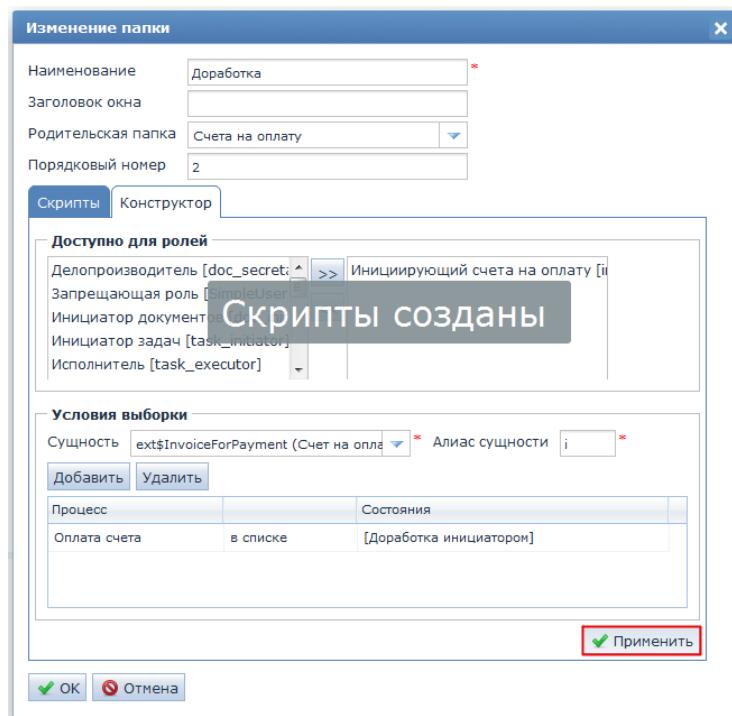


Рисунок 89. Окно добавления условий по процессу

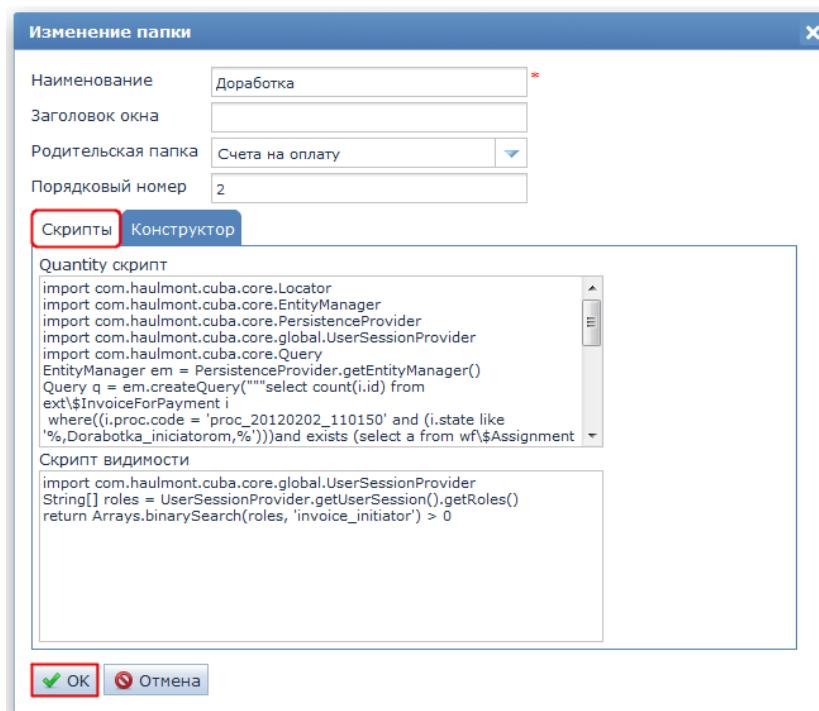
В отобразившемся окне из раскрывающегося списка выберите процесс **Оплата счета**. После того, как Вы выберите данный процесс, в панели **Состояния** отобразятся состояния данного процесса, которые были созданы в **Дизайне процессов ТЕЗИС**. Перенесите из этого списка на панель справа состояние **Доработка инициатором** с помощью кнопки **>>** и нажмите на кнопку **OK**.

Далее в окне **Изменение папки** нажмите на кнопку **Применить**. Система выдаст сообщение о том, что скрипты созданы.



**Рисунок 90. Окно редактирования папки Доработка**

В том, что скрипты созданы, можно убедиться, перейдя на вкладку **Скрипты**.



**Рисунок 91. Окно редактирования папки Доработка**

Нажмите на кнопку **OK**. Папка **Доработка** создана.

Создадим по аналогии папки **Утверждение**, **Согласование**, **Проверка** и **К оплате**.

Для создания папки **Утверждение** нажмите правой кнопкой мыши на папке **Счета на оплату**, далее в отобразившемся контекстном меню выберите пункт **Создать**. Перед Вами откроется окно, представленное на рисунке ниже.

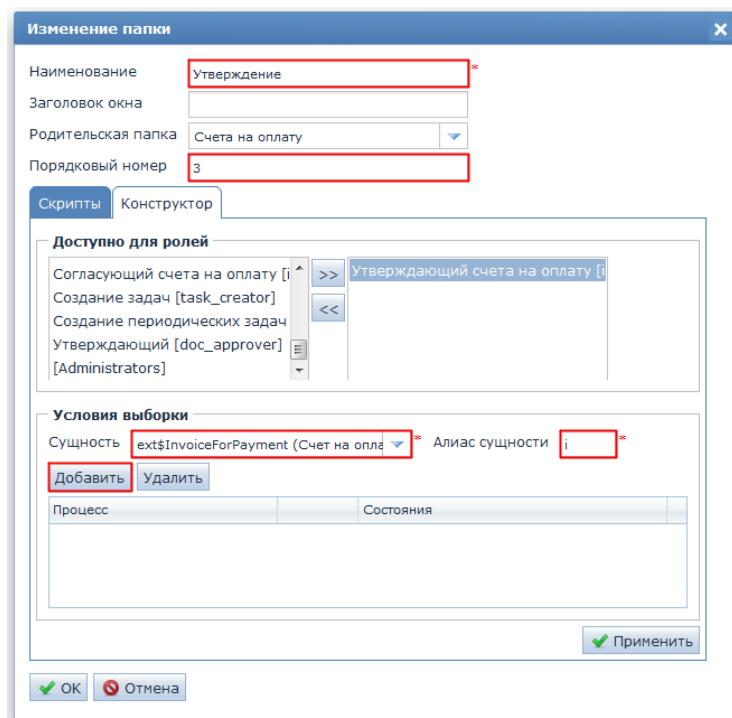


Рисунок 92. Окно редактирования папки Утверждение

В поле ввода **Наименование** введите название папки – **Утверждение**. В поле ввода **Порядковый номер** задается порядковый номер данной папки в списке папок. Далее перейдите на вкладку **Конструктор**. В панели **Доступно для ролей** из списка слева выберите роль **Утверждающий счета на оплату** путем нажатия на кнопку **>>**. В панели **Условия выборки** выберите сущность **ext\$InvoiceForPayment (Счет на оплату)** из списка предложенных сущностей и задайте алиас сущности.

Далее укажем, при каком условии в папку **Утверждение** будут поступать карточки счетов на оплату. Для этого нажмите на кнопку **Добавить**.

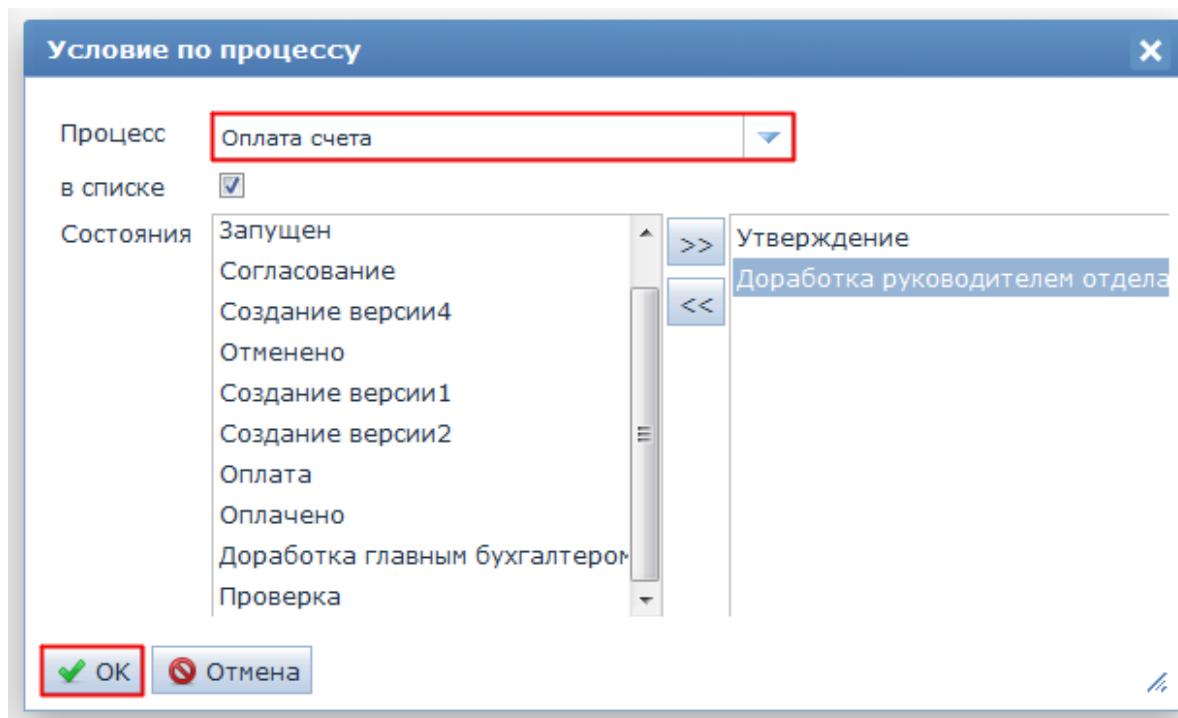
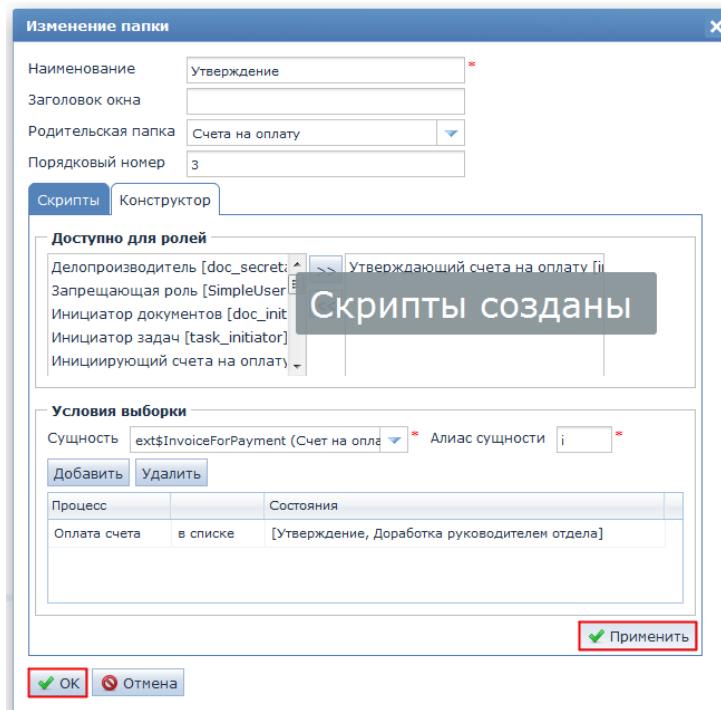


Рисунок 93. Окно добавления условий по процессу

В отобразившемся окне из раскрывающегося списка выберите процесс **Оплата счета**. После того, как Вы выберите данный процесс, в панели **Состояния** отобразятся состояния данного процесса, которые были созданы в **Дизайне процессов ТЕЗИС**. Перенесите из этого списка на панель справа состояния **Утверждение** и **Доработка руководителем отдела** с помощью кнопки **>>** и нажмите на кнопку **OK**.

Далее в окне **Изменение папки** нажмите на кнопку **Применить**. Система выдаст сообщение о том, что скрипты созданы.



**Рисунок 94. Окно редактирования папки Утверждение**

Нажмите на кнопку **OK**. Папка **Утверждение** создана.

Для создания папки **Согласование** нажмите правой кнопкой мыши на папке **Счета на оплату**, далее в отобразившемся контекстном меню выберите пункт **Создать**. Перед Вами откроется окно, представленное на рисунке ниже.

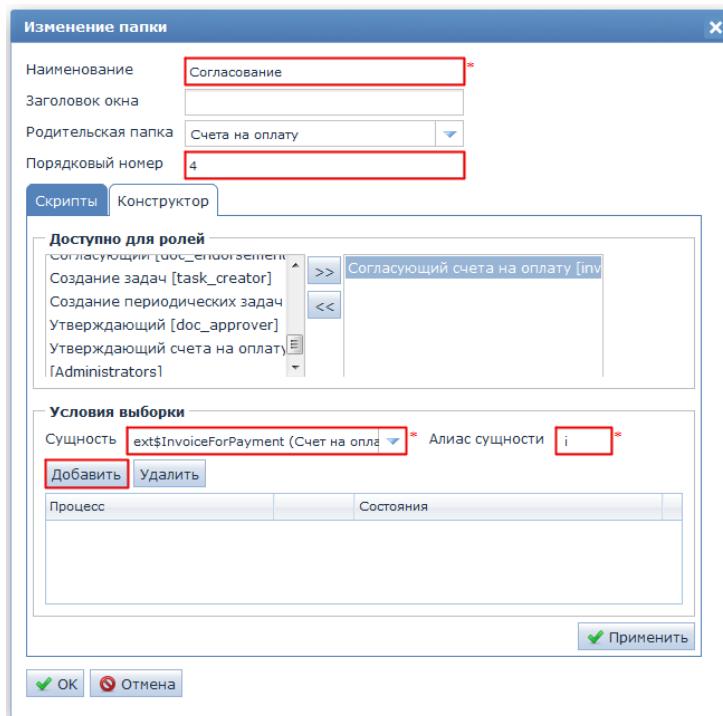


Рисунок 95. Окно редактирования папки Согласование

В поле ввода **Наименование** введите название папки – **Согласование**. В поле ввода **Порядковый номер** задается порядковый номер данной папки в списке папок. Далее перейдите на вкладку **Конструктор**. В панели **Доступно для ролей** из списка слева выберите роль **Согласующий счета на оплату** путем нажатия на кнопку **>>**. В панели **Условия выборки** выберите сущность **ext\$InvoiceForPayment** (**Счет на оплату**) из списка предложенных сущностей и задайте алиас сущности.

Далее укажем, при каком условии в папку **Согласование** будут поступать карточки счетов на оплату. Для этого нажмите на кнопку **Добавить**.

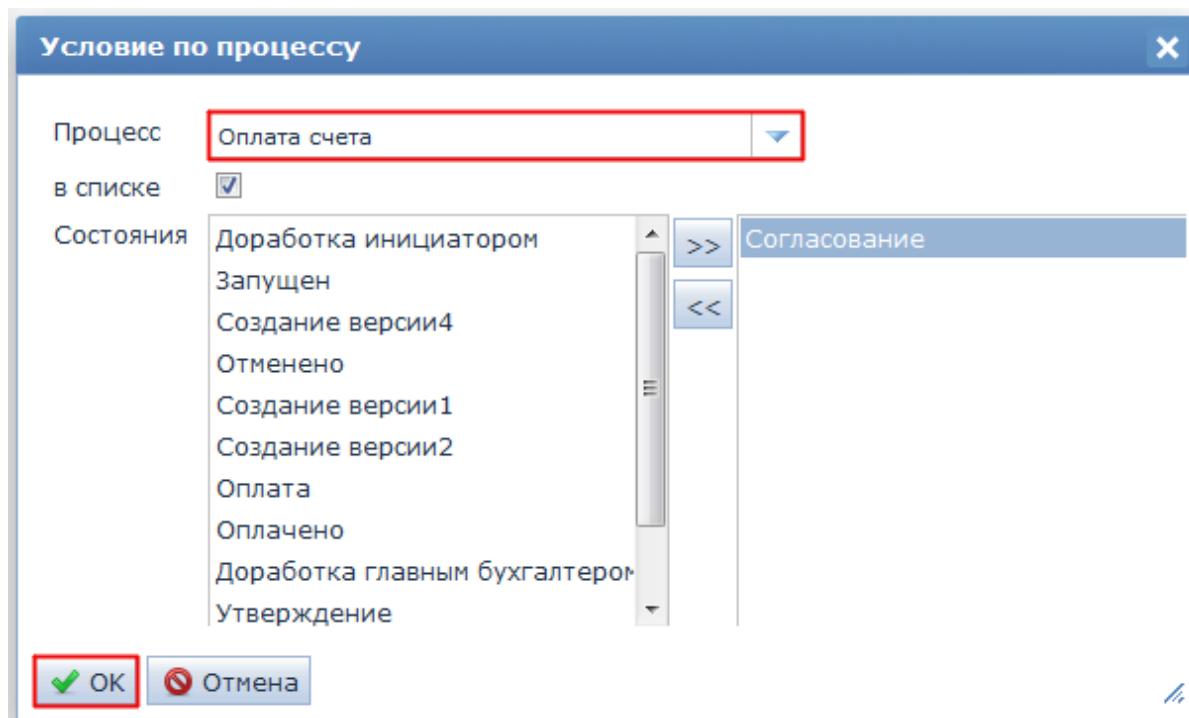
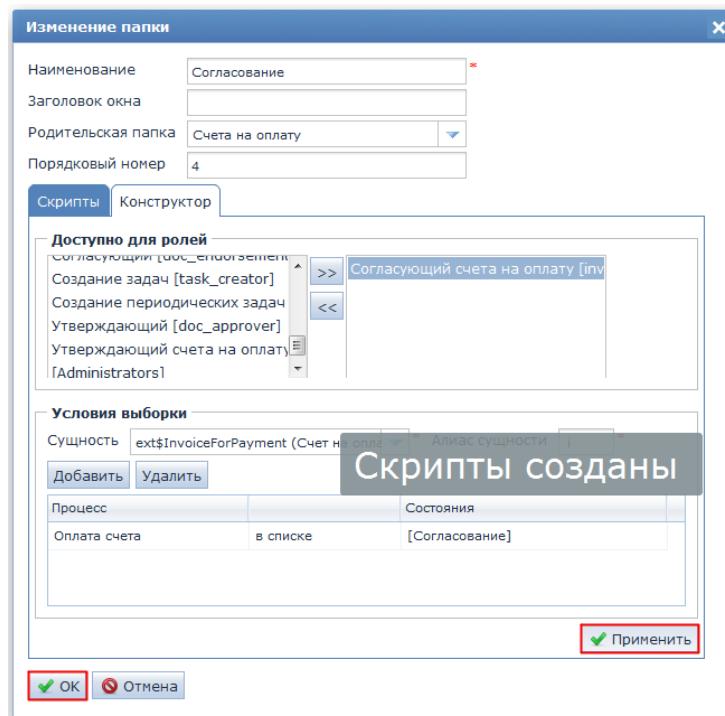


Рисунок 96. Окно добавления условий по процессу

В отобразившемся окне из раскрывающегося списка выберите процесс **Оплата счета**. После того, как Вы выберите данный процесс, в панели **Состояния** отобразятся состояния данного процесса, которые были созданы в **Дизайне процессов ТЕЗИС**. Перенесите из этого списка на панель справа состояние **Согласование** с помощью кнопки **>>** и нажмите на кнопку **OK**.

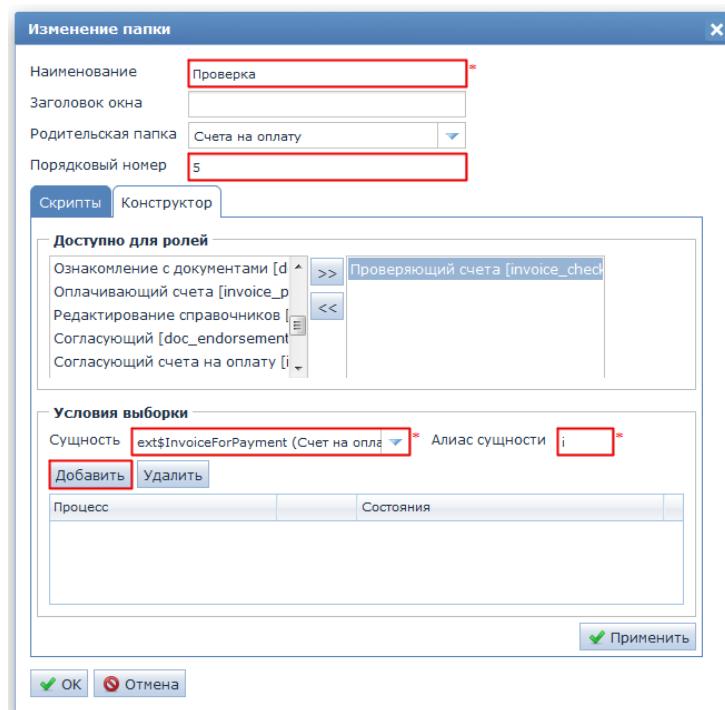
Далее в окне **Изменение папки** нажмите на кнопку **Применить**. Система выдаст сообщение о том, что скрипты созданы.



**Рисунок 97. Окно редактирования папки Согласование**

Нажмите на кнопку **OK**. Папка **Согласование** создана.

Для создания папки **Проверка** нажмите правой кнопкой мыши на папке **Счета на оплату**, далее в отобразившемся контекстном меню выберите пункт **Создать**. Перед Вами откроется окно, представленное на рисунке ниже.



**Рисунок 98. Окно редактирования папки Проверка**

В поле ввода **Наименование** введите название папки – **Проверка**. В поле ввода **Порядковый номер** задается порядковый номер данной папки в списке папок. Далее перейдите на вкладку **Конструктор**. В панели **Доступно для ролей** из списка слева выберите роль *Проверяющий счета на оплату* путем нажатия на кнопку **>>**. В панели **Условия выборки** выберите сущность *ext\$InvoiceForPayment* (*Счет на оплату*) из списка предложенных сущностей и задайте алиас сущности.

Далее укажем, при каком условии в папку **Проверка** будут поступать карточки счетов на оплату. Для этого нажмите на кнопку **Добавить**.

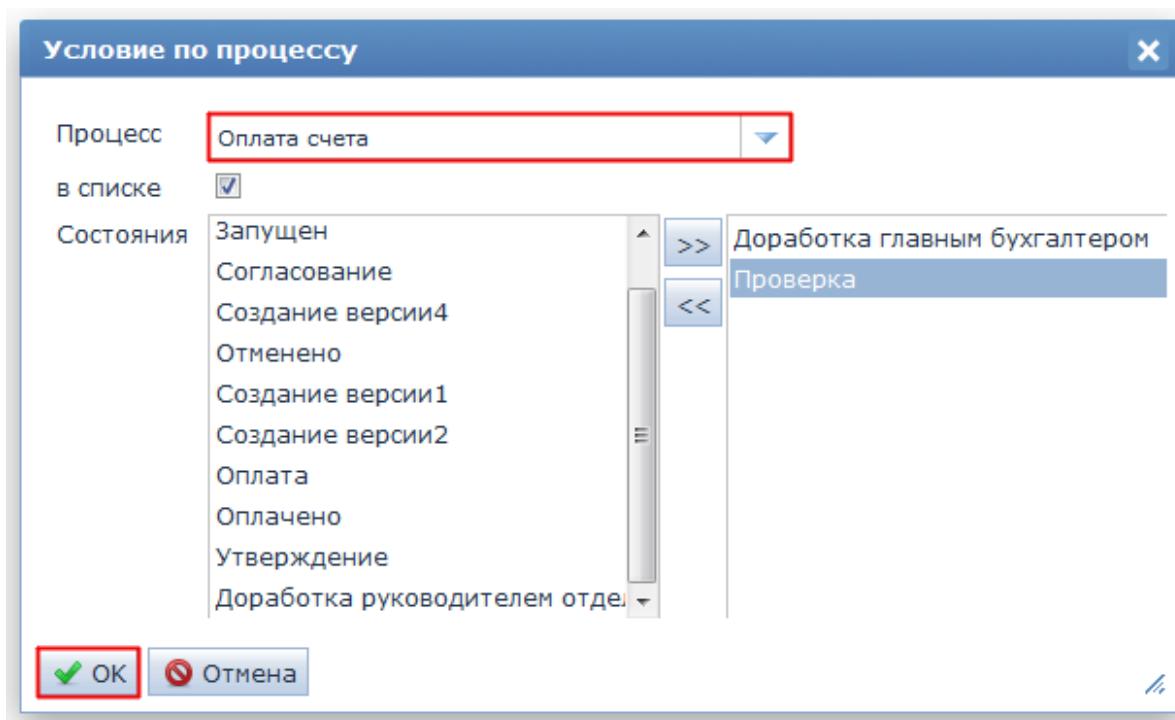
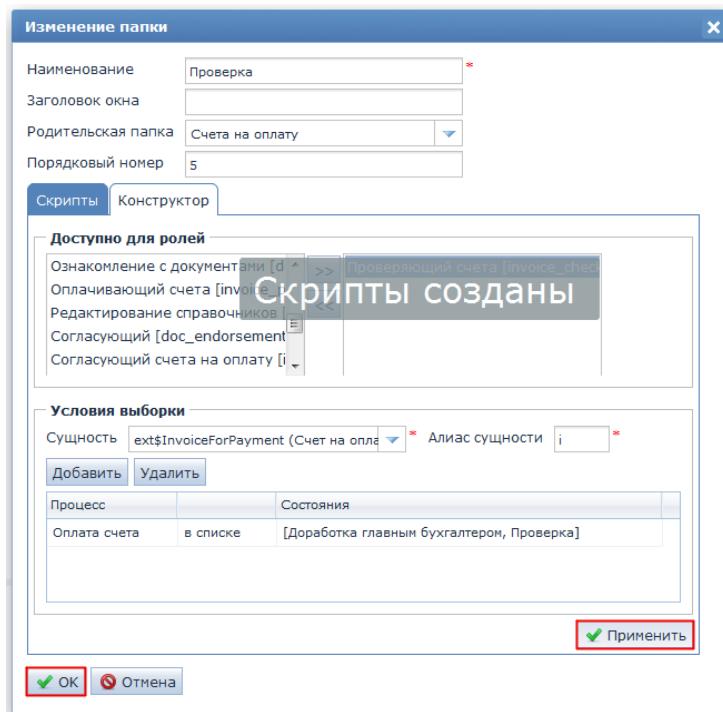


Рисунок 99. Окно добавления условий по процессу

В отобразившемся окне из раскрывающегося списка выберите процесс **Оплата счета**. После того, как Вы выберите данный процесс, в панели **Состояния** отобразятся состояния данного процесса, которые были созданы в **Дизайне процессов ТЕЗИС**. Перенесите из этого списка на панель справа состояния **Проверка** и **Доработка главным бухгалтером** с помощью кнопки **>>** и нажмите на кнопку **OK**.

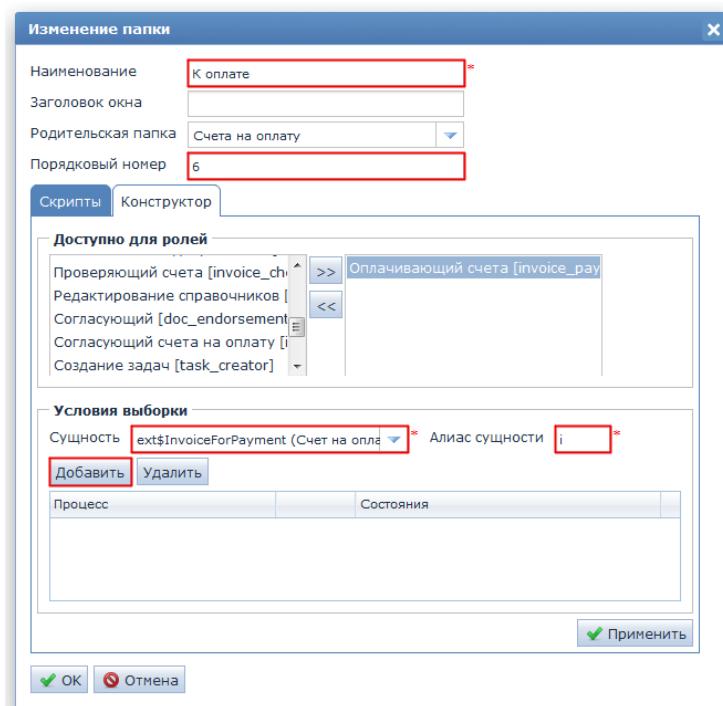
Далее в окне **Изменение папки** нажмите на кнопку **Применить**. Система выдаст сообщение о том, что скрипты созданы.



**Рисунок 100. Окно редактирования папки Проверка**

Нажмите на кнопку **OK**. Папка **Проверка** создана.

Для создания папки **К оплате** нажмите правой кнопкой мыши на папке **Счета на оплату**, далее в отобразившемся контекстном меню выберите пункт **Создать**. Перед Вами откроется окно, представленное на рисунке ниже.



**Рисунок 101. Окно редактирования папки К оплате**

В поле ввода **Наименование** введите название папки – **К оплате**. В поле ввода **Порядковый номер** задается порядковый номер данной папки в списке папок. Далее перейдите на вкладку **Конструктор**. В панели **Доступно для ролей** из списка слева выберите роль **Оплачивающий счета** путем нажатия на кнопку **>>**. В панели **Условия выборки** выберите сущность **ext\$InvoiceForPayment** (Счет на оплату) из списка предложенных сущностей и задайте алиас сущности.

Далее укажем, при каком условии в папку **К оплате** будут поступать карточки счетов на оплату. Для этого нажмите на кнопку **Добавить**.

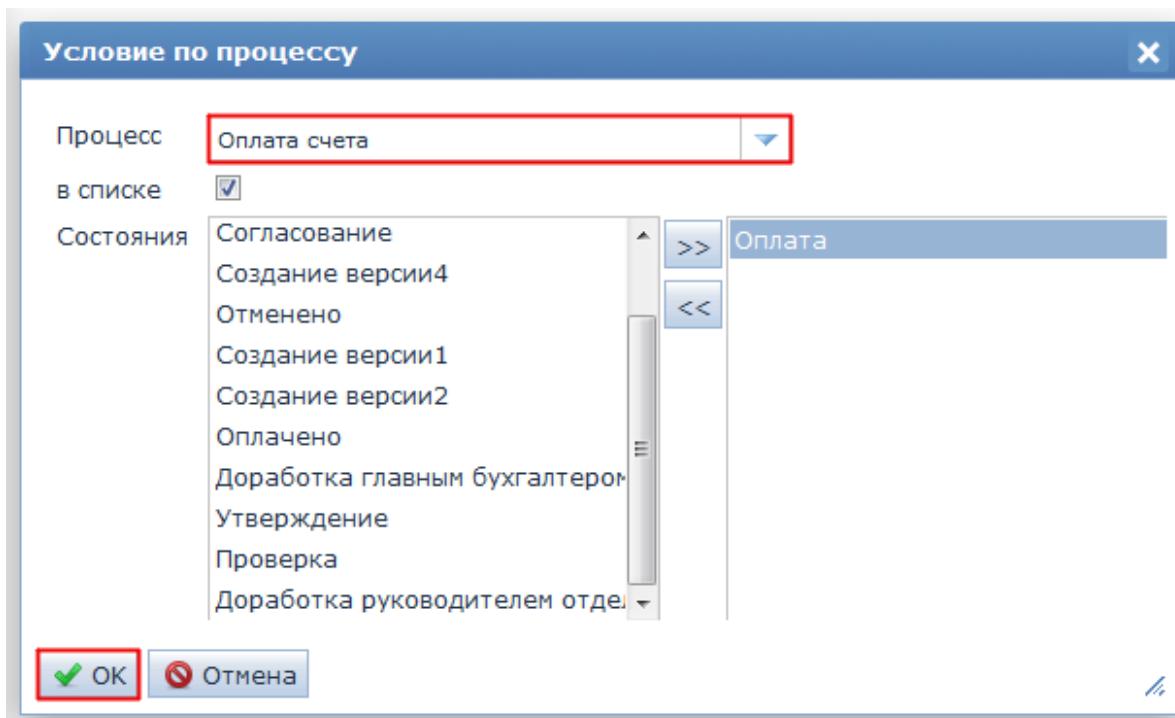
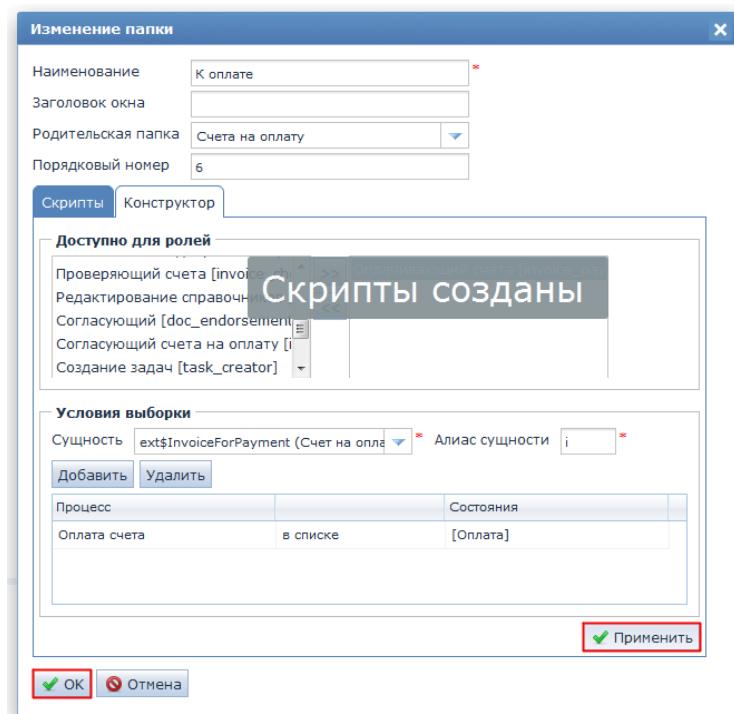


Рисунок 102. Окно добавления условий по процессу

В отобразившемся окне из раскрывающегося списка выберите процесс **Оплата счета**. После того, как Вы выберите данный процесс, в панели **Состояния** отобразятся состояния данного процесса, которые были созданы в **Дизайне процессов ТЕЗИС**. Перенесите из этого списка на панель справа состояние **Оплата** с помощью кнопки **>>** и нажмите на кнопку **OK**.

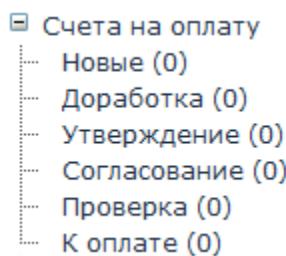
Далее в окне **Изменение папки** нажмите на кнопку **Применить**. Система выдаст сообщение о том, что скрипты созданы.



**Рисунок 103. Окно редактирования папки К оплате**

Нажмите на кнопку **OK**. Папка **К оплате** создана.

Вид папок приложения для процесса оплаты счета представлены на рисунке.



**Рисунок 104. Вид папок приложения**

## Приложение А. Часто задаваемые вопросы

### A.1. Как добавить фильтр в браузер справочника?

Для этого нужно добавить компонент *Generic Filter* в дескриптор экрана просмотра списка сущностей `budgetitem-browse.xml`:

```
<window
    xmlns="http://schemas.haulmont.com/cuba/5.2/window.xsd"
    class="com.haulmont.ext.web.ui.budgetitem.BudgetItemBrowser"
    messagesPack="com.haulmont.ext.web.ui.budgetitem"
    caption="msg://browserCaption"
    lookupComponent="budgetItemsTable"
    >
    ...
    <layout expandLayout="true">
        <vbox id="table-panel" expand="budgetItemsTable" spacing="true"
        height="100%">
            <filter id="genericFilter" datasource="budgetItemsDs"
            stylename="edit-area">
                <properties include=".*" exclude="" />
            </filter>
            ...
        </vbox>
    </layout>
</window>
```

### A.2. Как подключить полнотекстовый поиск созданной карточки?

Для этого нужно создать *дескриптор полнотекстового поиска ext-fts.xml* и написать в нем следующий код (пример приведен для сущности `InvoiceForPayment` (Счет на оплату)):

```
<fts-config>
    <entities>
        <entity class="com.haulmont.ext.core.entity.InvoiceForPayment">
            <include re=".*/>
            <include name="attachments.file"/>
            <include name="assignments.attachments.file"/>

            <exclude name="jbpmProcessId"/>
            <exclude name="parentCard"/>
            <exclude name="subCards"/>
        </entity>
    </entities>
</fts-config>
```

Далее убедитесь, чтобы в файле `ext-app.properties` был указан дескриптор:

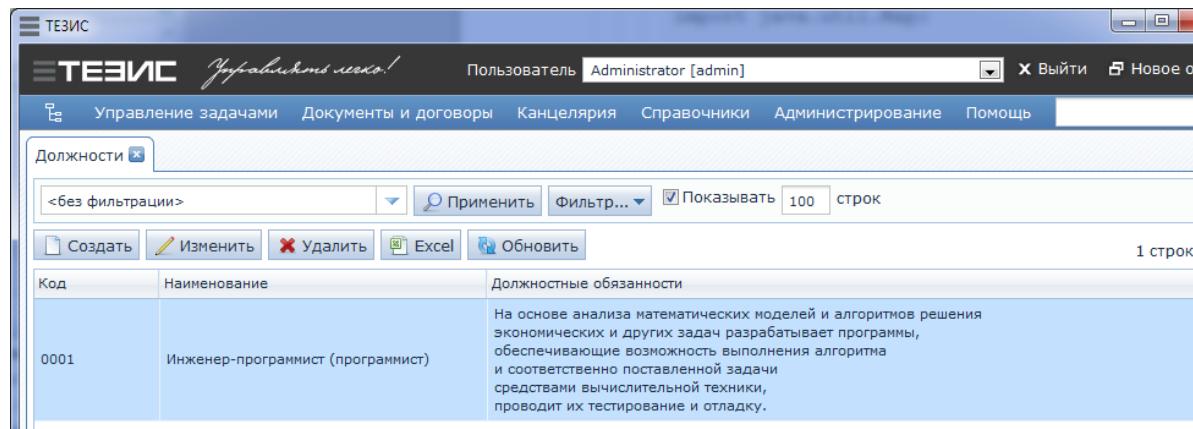
```
cuba.ftsConfig=cuba-fts.xml thesis-fts.xml ext-fts.xml
```

### A.3. Как подключить историю открытых экранов карточки (меню Помощь -> История)

Откройте файл свойств ext-web-app.properties и напишите следующее (пример приведен для сущности InvoiceForPayment (Счет оплату) ):

```
cuba.screenIdsToSaveHistory=tm$Task.edit,df$SimpleDoc.edit,\n        df$Contract.edit,tm$TaskPattern.edit,\n        tm$TaskGroupPattern.edit,\n        tm$TaskType.edit,df$DocType.edit,\n        df$DocKind.edit,df$OfficeFileNomenclature.edit,\n        df$OfficeFile.edit,df$DocReceivingMethod.edit,\n        tm$Priority.edit,tm$ProjectGroup.edit,\n        df$Category.edit,wf$UserGroup.edit,\n        wf$AttachTypes.edit,df$Organization.edit,\n        df$Department.edit,df$Employee.edit,\n        df$Company.edit,df$Individual.edit,\n        df$Currency.edit,df$Bank.edit,\n        df$BankRegion.edit,sec$User.edit,sec$Group.edit,\n        sec$Role.edit,tm$ScheduleTask.edit,\n        ext$InvoiceForPayment.edit
```

### A.4. Как корректно выводить многострочный текст в таблице?



The screenshot shows a software interface titled 'ТЕЗИС Управление задачами'. The main window displays a table with three columns: 'Код', 'Наименование', and 'Должностные обязанности'. The 'Код' column contains '0001', the 'Наименование' column contains 'Инженер-программист (программист)', and the 'Должностные обязанности' column contains a multi-line text describing the responsibilities of the position.

Код	Наименование	Должностные обязанности
0001	Инженер-программист (программист)	На основе анализа математических моделей и алгоритмов решения экономических и других задач разрабатывает программы, обеспечивающие возможность выполнения алгоритма и соответственно поставленной задачи средствами вычислительной техники, проводит их тестирование и отладку.

**Рисунок 105. Вид таблицы с многострочным текстом**

Такая ситуация возникает в справочнике должностей.

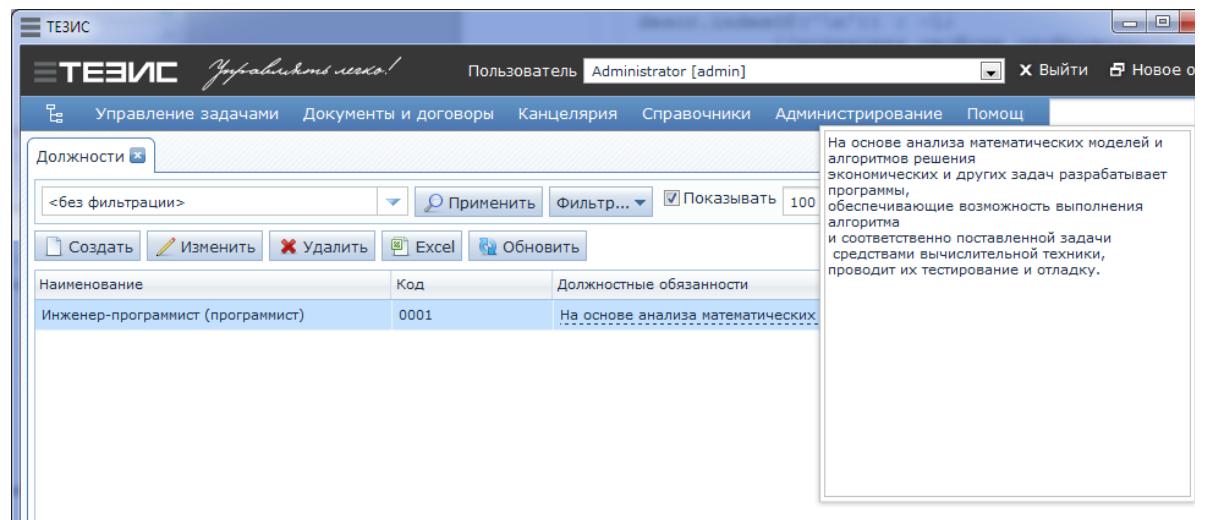
Зададим собственное представление данных в колонке **Должностные обязанности**, то есть создадим генерируемую колонку. Для этого в методе init() добавьте вызов метода addCommentColumn(), программный код которого приведен ниже:

```
protected void addCommentColumn() {\n    MetaPropertyPath pp = postsDs.getMetaClass().getPropertyPath("jobDuties");\n    com.vaadin.ui.Table vTable = (com.vaadin.ui.Table) WebComponentsHelper\n        .unwrap(postsTable);
```

```

vTable.removeGeneratedColumn(pp);
vTable.addColumn(
    pp,
    new com.vaadin.ui.Table.ColumnGenerator() {
        public com.vaadin.ui.Component generateCell(com.vaadin.ui.Table source,
                                                      Object itemId, Object columnId) {
            UUID uuid = (UUID) itemId;
            com.vaadin.ui.Component component;
            Post post = postsDs.getItem(uuid);
            String descr = post.getJobDuties();
            int enterIdx = descr != null ?
                (descr.length() > 40 ? 40 : descr.indexOf('\n')) : -1;
            //установка свойств отображаемого компонента
            if (enterIdx != -1) {
                com.vaadin.ui.TextField content =
                    new com.vaadin.ui.TextField(null, descr);
                content.setReadOnly(true);
                content.setWidth("300px");
                content.setHeight("300px");
                component = new com.vaadin.ui.PopupView("<span>" +
                    + descr.substring(0, enterIdx) + "...</span>", content);
                component.addStyleName("longtext");
            } else {
                component = new com.vaadin.ui.Label(descr == null ? "" : descr);
            }
            component.setWidth("-1px");
            return component;
        }
    });
}
    
```

Пересоберите проект и откройте справочник должностей. Убедитесь, многострочный текст отображается корректно.



**Рисунок 106. Вид таблицы с многострочным текстом**

## A.5. Как пересобрать проект?

Выполните следующие действия:

1. Запустите командную строку в *рабочем каталоге* .
2. В командной строке введите команду

```
gradle restart
```

#### A.6. Как посмотреть журналы сообщений сервера приложений Tomcat?

В первую очередь проверьте, что сервер приложений Tomcat установлен в *рабочем каталоге* . Информация о том, как это сделать, находится в главе **Создание и запуск проекта расширения для системы ТЕЗИС** .

Если сервер установлен и если проект был хотя бы раз собран, то в рабочем каталоге должна быть папка `build/tomcat/logs` . Для просмотра журнала сообщений откройте в этой папке файл `app.log` .

#### A.7. Как подключить программу "ТЕЗИС: Оповещения" к своему проекту?

Откройте файл `web.xml` модуля `web` и добавьте в него следующий код:

```
<servlet>
    <servlet-name>spring-ws</servlet-name>
    <servlet-class>
        org.springframework.ws.transport.http.MessageDispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>transformWsdlLocations</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/ws/ws-spring.xml</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>spring-ws</servlet-name>
    <url-pattern>/ws/*</url-pattern>
</servlet-mapping>
```

#### A.8. Почему не отправляются уведомления процесса на почту\в систему?

Скорее всего, проблема с матрицей оповещений. Перейдите к пункту меню **Администрирование -> Дизайн процессов** . В отобразившемся окне

таблице найдите процесс, для которого не отображаются оповещения и откро матрицу оповещений данного процесса, нажав ссылку **Показать** в колон **Матрица оповещений**.

Наименование	Дата создания	Матрица оповещений	Дата компиляции
Оплата счета	02.02.2012 11:01	<a href="#">Показать</a>	10.02.2012 10:57

Рисунок 107. Загрузка матрицы оповещений

Откройте выгруженную матрицу оповещений и перейдите на вкладку **Mail**, если оповещения не приходят на почту, или на вкладку **Tray**, если оповещения приходят в систему или в **"ТЕЗИС: Оповещения"**.

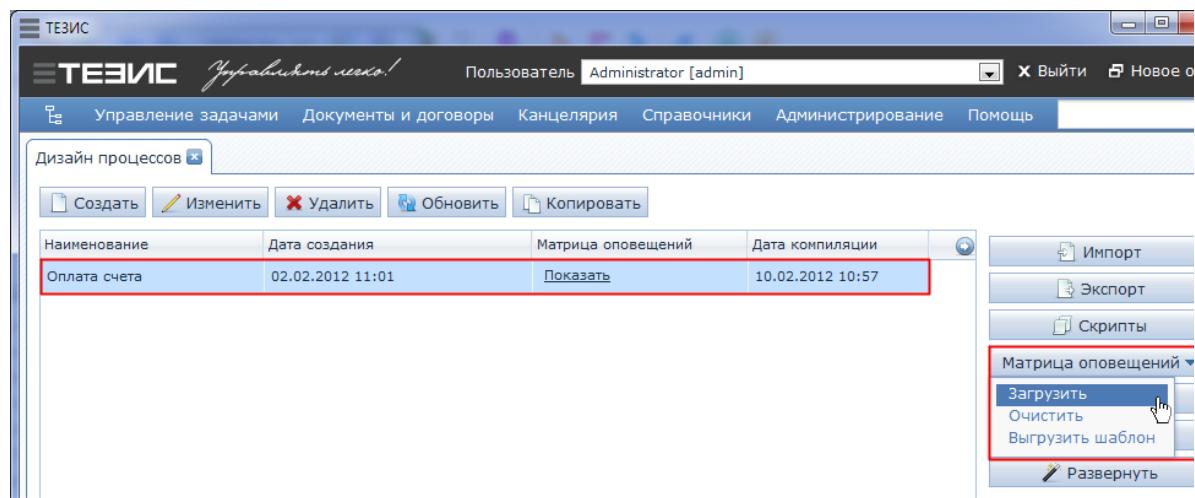
	A	B	C	D	E	F	G	H
1								
2								
3		Начало.Утверждение	Утверждение.Согласование	Утверждение.Доработка инициатором	Утверждение.Конец1	Согласование.Проверка	Согласование.Доработка инициатором	Согласование.Команды
4	Руководитель отдела	ACTION					ACTION	SIMPLE
5	Финансист		ACTION					
6	Главный бухгалтер				ACTION			
7	Бухгалтер							
8	Инициатор			ACTION	SIMPLE			SIMPLE
9								

Рисунок 108. Матрица оповещений

Таблицы на вкладках **Mail** и **Tray** должны быть обязательно заполнены.

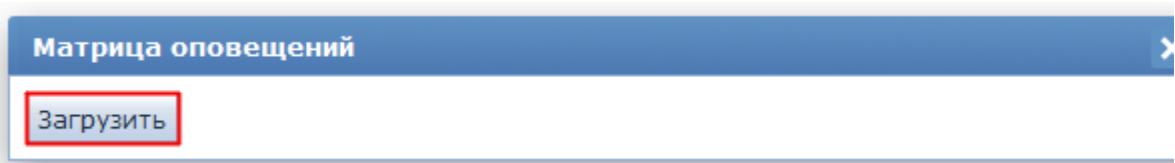
Если таблицы не заполнены, заполните их, сохраните файл матрицы оповещений и закройте его.

Далее в системе выделите строку с дизайном процесса, для которого будет выгружена матрица оповещений, и в панели справа нажмите на кнопку **Матрица оповещений**. В выпадающем списке выберите пункт **Загрузить**.



**Рисунок 109. Загрузка матрицы оповещений**

Далее в отобразившемся окне нажмите на кнопку **Загрузить** и укажите путь сохраненной ранее матрице оповещений.



**Рисунок 110. Окно загрузки матрицы оповещений**

После загрузки матрицы оповещений скомпилируйте и разверните процесс, создавая его заново. Подробные инструкции по обработке дизайна процесса содержатся в главе 2.11.5 *Руководства администратора* системы **ТЕЗИС**.

## Основные определения и понятия

### Б

#### Браузер

Экранная форма, на которой размещается таблица со списком сущностей, а также кнопки создания, редактирования, удаления сущности.

### В

#### Внедрение зависимости

Механизм для получения ссылок на используемые объекты. Для получения более подробной информации обратитесь к статье **Википедии**  
[http://ru.wikipedia.org/wiki/Внедрение\\_зависимости](http://ru.wikipedia.org/wiki/Внедрение_зависимости)

### Г

#### Главный пакет сообщений

*Пакет локализованных сообщений* , содержащий заголовки пунктов меню, названия кнопок и других элементов интерфейса. Располагается в пакете com.haulmont.ext.web модуля web .

### Д

#### Дескриптор FTS

Дескриптор *FTS* представляет собой XML файл, содержащий настройки механизма полнотекстового поиска. Элементы дескриптора:

- ***include*** . Включение содержимого другого дескриптора.  
Атрибут *file* – путь к включаемому файлу относительно каталога конфигурации.
- ***entities*** . Список сущностей, подлежащих индексированию и поиску.
  - ***entity*** . Описание индексируемой сущности.  
Атрибуты:
    - *class* – java класс сущности;
    - *show* (true/false) – должна ли данная сущность показываться в результатах поиска

самостоятельно. `False` используется для сущностей-связей, которые нужны, например, для связи файлов и карточек;

- `include` – включить атрибут сущности в индекс.

Атрибуты:

- `re` – регулярное выражение для отбора атрибутов по имени. Отбираются только атрибуты следующих типов: строка, число, дата, перечисление;

- `name` – имя атрибута, может быть путем по ссылочным атрибутам. Тип не проверяется, однако если имя является путем, то конечный атрибут должен быть сущностью, а не простым типом (атрибут простого типа не имеет здесь смысла, он должен индексироваться в своей сущности);

- `exclude` – исключить ранее включенный атрибут.

Возможные атрибуты такие же как в элементе `include`;

- `searchables` – Groovy -скрипт для добавления в очередь на индексирование произвольных сущностей, связанных с измененной. Скрипт оперирует двумя переменными:

`searchables` – список сущностей;

`entity` – текущая сущность, помещаемая в очередь автоматически;

- `searchableIf` – Groovy -скрипт для ограничения помещения в очередь некоторых экземпляров индексируемой сущности. Скрипт оперирует переменной `entity` (текущая сущность, которую нужно проверить).

Пример дескриптора:

```

<?xml version="1.0" encoding="UTF-8"?>
<fts-config>
    <include file="cuba/fts-config.xml"/>
    <entities>
        <entity class="com.haulmont.workflow.core.entity.CardAttachment">
            show="false"
            <include re=".*"/>
            <exclude name="card"/>
            <searchables>
                searchables.add(entity.card)
            </searchables>
        </entity>

        <entity class="com.haulmont.workflow.core.entity.AssignmentAttachment">
            show="false"
            <include re=".*"/>
            <exclude name="assignment"/>
            <searchables>
                searchables.add(entity.assignment.card)
            </searchables>
        </entity>

        <entity class="com.haulmont.thesis.core.entityaskman.core.entity.ProjectGroup">
            <include name="name"/>
        </entity>

        <entity class="com.haulmont.thesis.core.entityre.entity.Project">
            <include re=".*"/>
        </entity>

        <entity class="com.haulmont.thesis.core.entity.Task">
            <include re=".*"/>
            <include name="attachments.file"/>
            <include name="assignments.attachments.file"/>
            <exclude name="parentCard"/>
            <exclude name="subCards"/>
            <exclude name="timeUnit"/>
        </entity>

        <entity class="com.haulmont.thesis.core.entity.Contract">
            <include re=".*"/>
            <include name="attachments.file"/>
            <include name="assignments.attachments.file"/>
            <exclude name="parentCard"/>
            <exclude name="subCards"/>
            <searchableIf>
                entity.versionOf == null
            </searchableIf>
        </entity>
    </entities>
</fts-config>

```

**И**

## Источник данных

Предназначен для реализации связанных с данными компонентов.

Существуют следующие интерфейсы источников данных:

- Datasource – предназначен для работы с одним экземпляром

сущности.

- `RuntimePropsDatasource` – предназначен для работы с динамическими свойствами сущностей.
- `CollectionDatasource` – предназначен для работы с коллекцией сущностей
  - `HierarchicalDatasource` – предназначен для работы с компонентами `Tree`, `TreeTable`.
  - `GroupDatasource` – предназначен для работы с компонентом `GroupTable`.

Как правило, источники данных объявляются декларативно в секции `dsContext` дескриптора экрана .

## K

### Контроллер экрана

Java или Groovy класс, в котором можно реализовывать бизнес-логику и управлять поведением компонентов, описанных в `xml-дескрипторе`, с помощью различных событий. Контроллер должен быть унаследован от одного из следующих базовых классов:

- `AbstractFrame` – реализует интерфейс `IFrame` и предназначен для реализации фреймов – многократно используемых компонентов экранов.
- `AbstractWindow` – реализует интерфейс `Window` и предназначен для реализации любых экранов.
- `AbstractLookup` – реализует интерфейс `Lookup` и предназначен для реализации `браузеров` с возможностью выбора элемента списка для использования его в вызывающем экране.
- `AbstractEditor` – реализует интерфейс `Editor` и предназначен для реализации экранов редактирования экземпляра сущности.

Основной метод контроллера – `init()`. Этот метод вызывается после

создания класса окна и всего дерева компонентов, описанного XML-дескриптором. Контроллер может в этом методе произвести инициализацию экрана перед его открытием (например, создать обработчики нажатий на кнопки и пр.).

В метод `init()` из вызывающего кода передается коллекция параметров, состоящая из пар «ключ – значение». Вызывающим кодом может быть как контроллер другого экрана, так и пункт главного меню, либо прямая ссылка на экран извне приложения. Каждый параметр передается в коллекции в двух экземплярах – с простым именем и с тем же именем с префиксом `param$`.

Класс контроллера должен быть зарегистрирован в XML-дескрипторе экрана в атрибуте `class` корневого элемента.

## H

### Нумератор

Системный объект, позволяющий организовать сквозную нумерацию документов разных видов.

Скрипты нумераторов пишутся на языке Groovy. Для генерации последовательностей рекомендуется использовать интерфейс `UniqueNumbers`.

За уникальность нумерации отвечает метод `getNextNumber("numeratorName")`. Если название "numeratorName" будет использоваться в другом нумераторе, то получится, что два нумератора будут пользоваться одной последовательностью.

## P

### Пакет локализованных сообщений

Содержит файлы локализованных сообщений для задания названий сущностей, атрибутов сущностей, папок, заголовков вкладок и так далее. Правила создания файлов локализованных сообщений:

- Файл по умолчанию имеет имя `messages.properties`, для конкретного языка – `messages_ru.properties` и т.д. При

отсутствии файла конкретного языка или при отсутствии искомого ключа в файле конкретного языка будет использовано сообщение из файла по умолчанию.

- Кодировка UTF-8 .
- Можно включать другие файлы сообщений с помощью ключа @include . При этом если некоторый ключ встречается и во включаемом файле, и в текущем, будет использовано сообщение из текущего ( переопределение ). Пример включения файла:

```
@include=com.haulmont.cuba.web

actions.Create=Create
actions.Edit>Edit
...
```

- Полным именем пакета сообщений считается имя Java-пакета, содержащего языковые файлы. Например именем пакета сообщений, расположенного в /com/haulmont/cuba/web/app/ui/security/ user/messages.properties является строка com.haulmont.cuba.web.app.ui.security.user

### Папки поиска

Папки данной категории могут быть созданы и настроены пользователем (за исключением подкатегории "Системные"). Подкатегория "Системные" содержит предустановленные папки поиска, которые доступны всем пользователям системы.

Папки поиска обладают следующими особенностями:

- открывают экраны с generic-фильтром ;
- создаются пользователем из любого открытого экрана с generic-фильтром;
- пользователь может изменить иерархию, название, либо переопределить фильтр поиска.

### Папки приложения

Папки данной категории содержат задачи, по которым требуются действия пользователя.

Обладают следующими особенностями:

- открывают предопределенные экраны с **фильтром** или без;
- набор папок зависит от ролей пользователя;
- пользователь не может изменить настройки папок;
- в заголовке папки может отображаться текущее количество входящих в нее записей;
- периодически обновляются по таймеру.

### Представление

Механизм представлений обеспечивает извлечение из БД и передачу клиенту графов сущностей, ограниченных в глубину и/или по атрибутам.

Представление всегда должно быть задано при запросе данных у *middleware*. Получив выборку данных по представлению, клиентский код может быть уверен, что все запрошенные атрибуты и связанные сущности получены, и их можно использовать без опаски получить *NullPointerException* или исключение типа *LazyInitializationException*. Представление решает и обратную задачу – ненужные атрибуты не извлекаются из БД, что снижает нагрузку в случае "широких" сущностей (содержащих большое число атрибутов).

Для каждой сущности по умолчанию доступны два типа представления с именами `_local` и `_minimal`

- `_local` определяет все локальные атрибуты сущности (т.е. все, кроме ссылок на другие сущности);
- `_minimal` определяет атрибуты, входящие в `InstanceName`. Если `InstanceName` не задано, данное представление определяет только системные атрибуты (`id`, `createTs` и пр.).

Представления нужно определять в файле `ext-views.xml` модуля `core`.

### Внимание

Имя представления должно быть уникально в пределах сущности. При этом рекомендуется давать более "описательные" имена. Например, не "browse", а "nameOfEntityBrowse". Это упрощает поиск XML-описателей представлений по имени.

Пример представления для сущности `User`, которое должно выбирать все простые поля и ссылку на группу:

```
<view entity="sec$User" name="user.browse" extends="_local">
    <property name="group" view="_local"/>
</view>
```

### Процесс

Под процессом будем понимать последовательность выполнения действий, переходящую от одного человека к другому, а для больших процессов – от одного отдела к другому. Процесс всегда имеет начальную и завершающую стадию, а также алгоритм действий.

## P

### Рабочий каталог

Каталог файловой системы, в котором содержится Ваш проект. Рабочий каталог, если он был создан, должен содержать скрипты сборки `build.gradle`, `settings.gradle` и проектные файлы **IntelliJ IDEA**. Информация о том, как создать такой каталог, содержится в главе Создание и запуск проекта расширения для системы **ТЕЗИС**.

## B

### ButtonsPanel

Унифицирует использование и размещение кнопок для управления данными в таблице.

XML-имя компонента: `buttonsPanel`.

Атрибуты `buttonsPanel`:

- `id`

- visible
- align
- expandable
- width
- height

## D

### Datasource

См. [Источник данных](#).

### DateField

Поле для редактирования даты и времени.

XML-имя компонента: `dateField`.

Атрибуты `dateField`:

- id
- stylename
- height
- width
- visible
- enable
- expandable
- datasource
- property
- caption
- description
- editable
- required
- requiredMessage
- `dateFormat` – формат представления даты/времени по правилам

SimpleDateFormat. Значением атрибута может быть либо строка формата, либо ключ в пакете сообщений (если строка начинается с msg://)

- *resolution* – точность представления даты/времени (день, час и т.д.). Значение атрибута должно соответствовать перечислению `DateField.Resolution` – MSEC, SEC, MIN, HOUR, DAY, MONTH, YEAR. По умолчанию точность определяется по аннотации `javax.persistence.Temporal` соответствующего атрибута сущности.

Если *resolution*=“DAY” и не указан атрибут `dateFormat`, то в качестве формата будет взят формат, указанный в главном пакете сообщений с ключом `dateFormat`.

Если *resolution*=“MIN” и не указан атрибут `dateFormat`, то в качестве формата будет взят формат, указанный в главном пакете сообщений с ключом `dateTimeFormat`.

Элементы `dateField`:

- `validator`

## E

### **EntityManager**

Служит для извлечения сущностей из БД и сохранения их в БД. Может быть получен только в слое `middleware` с помощью `PersistenceProvider`.

## F

### **Full Text Search**

Полнотекстовый поиск предоставляет возможность неструктурированного поиска по всем атрибутам сущностей и по содержимому вложенных файлов.

FTS содержит два взаимосвязанных механизма – индексирование и поиск.

**Индексирование.** Изменения сущностей отслеживаются на уровне `EntityManager`'а. Если изменился атрибут сущности, подлежащий индексированию, информация о сущности записывается в таблицу `SYS_FTS_QUEUE` с помощью `FtsSenderBean`. Отдельный

асинхронный процесс извлекает ссылки на изменившиеся сущности из очереди, загружает сущности и индексирует их.

Под индексируемыми атрибутами понимаются атрибуты данной сущности и ссылающихся на нее сущностей, объявленные в [дескрипторе FTS](#).

**Поиск.** Поиск ведется по совпадению искомой строки с началом слов индексированных данных. При включенном FTS главное окно приложения содержит поле поиска. Поиск производится с помощью `FtsServiceBean` и результаты отображаются в окне `search-results.xml`.

## G

### Generic Filter

Служит для создания и применения произвольных фильтров для `CollectionDatasources` во время работы приложения.

### GridLayout

Контейнер, располагающий компоненты по сетке.

XML-имя компонента: `grid`.

Атрибуты `grid`:

- `id`
- `visible`
- `spacing`
- `margin`
- `width`
- `height`
- `expandable`
- `align`
- `column` – необязательный элемент, декларирует атрибуты колонок сетки. Атрибуты:
  - `flex` – необязательный атрибут, задает соотношение ширин колонок.

- *columns* – обязательный элемент, содержит последовательность *column*. Атрибуты:
  - *count* – необязательный атрибут, задает количество колонок, если не заданы элементы *column*.
- *rows* – обязательный элемент, содержит последовательность *row*.
- *row* – обязательный элемент, контейнер для содержимого ряда.
  - *flex* – необязательный атрибут, задает соотношение высот рядов сетки.
  - *visible*
  - *spacing*
  - *align*

Пример использования компонента в **XML-дескрипторе** :

```
<grid spacing="true" expandable="false">
    <columns>
        <column flex="1"/>
        <column flex="1"/>
    </columns>
    <rows>
        <row>
            <label value="msg://name"/>
            <textField id="name" datasource="fileDs"
                property="name"
                width="200px"/>
        </row>
        <row>
            <label value="msg://extension"/>
            <label id="extension" datasource="fileDs"
                property="extension"/>
        </row>
    </rows>
</grid>
```

### **GroupBoxLayout**

Контейнер, позволяющий выделить рамкой группу объектов, задать им общий заголовок и описание. Кроме того, он умеет сворачивать свое содержимое.

XML-имя компонента: **groupBox** .

Атрибуты **groupBox** :

- *id*
- *styleName*

- *height*
- *width*
- *visible*
- *enable*
- *expandable*
- *collapsible* – необязательный атрибут. Может принимать значение *true*, в этом случае компонент может сворачивать свое содержимое, или значение *false*.
- *collapsed* – необязательный атрибут, принимающий значение *true* или *false*. В значении *true* компонент будет свернут по-умолчанию.

Элементы *groupBox* :

- *caption* – необязательный элемент, его атрибут *label* содержит заголовок компонента.
- *description* – необязательный элемент, его атрибут *label* содержит описание.
- обязательный элемент – контейнер, содержащий вложенные компоненты (vbox, hbox, grid, flowbox).

Пример использования компонента в *XML-дескрипторе* :

```
<groupBox collapsable="true" stylename="edit-area">
    <caption label="msg://Caption"/>
    <description label="msg://Description"/>
    <vbox expandable="false">
        <twinColumn id="twinColGroups" optionsDatasource="groups"
captionProperty="name"
            expandable="false" rows="10" columns="15"/>
        <button enable="false" caption="msg://Button"/>
    </vbox>
</groupBox>
```

L

## **LookupField**

Компонент, реализующий выпадающий список. Он реализует фильтрацию значений в реальном времени (по мере ввода имени опции пользователем) и постраничный вывод доступных опций.

XML-имя компонента: *lookupField* .

Атрибуты *lookupField* :

- *id*
- *stylename*
- *height*
- *width*
- *visible*
- *enable*
- *expandable*
- *datasource*
- *property*
- *caption*
- *editable*
- *required*
- *requiredMessage*
- *optionsDatasource*
- *captionProperty*
- *nullName* – идентификатор опции, выбор которой будет равносителен установке значения в *null*.
- *filterMode* – тип фильтрации опций:
  - *NO* – нет фильтрации.
  - *STARTS\_WITH* – по началу фразы.
  - *CONTAINS* – по любому вхождению.
- *validator*

Пример использования компонента в *XML-дескрипторе* :

```
<lookupField id="docClass" datasource="docType" property="docClass"
             optionsDatasource="docClasses" captionProperty="name"
             required="true" width="300px"/>
```

## Managed Beans

Внутренние компоненты *middleware*, содержащие бизнес-логику приложения. Опционально могут иметь JMX интерфейс для управления во время работы приложения. В этом случае они должны быть созданы по правилам реализации MBeans.

### MBeans

Программные компоненты, реализующие бизнес-логику и имеющие JMX-интерфейс. Как правило, имеют внутреннее состояние (например, кэш, конфигурационные данные или статистику), к которому нужно обеспечить доступ через JMX. MBean является расширенным вариантом *Managed Bean*.

### Middleware

Средний слой приложения, содержит бизнес-логику и предоставляет общий интерфейс для верхних слоев приложения.

### Middleware Services

Middleware Services предоставляют интерфейс для вызова бизнес-логики клиентами и образуют границу *middleware*. Клиентский код не может обратиться ни к каким компонентам ядра middleware, только к интерфейсам сервисов. Сервисы могут содержать бизнес-логику внутри себя, либо делегировать выполнение *Managed Beans*.

## P

### PickerField

*PickerField* позволяет отображать экземпляр сущности в текстовом поле и выполнять действия нажатием на кнопки справа.



Рисунок 111. Вид компонента PickerField

XML-имя компонента: *pickerField*.

Атрибуты и элементы *pickerField*:

- id

- `stylename`
- `height`
- `width`
- `visible`
- `enable`
- `expandable`
- `datasource`
- `property`
- `caption`
- `editable`
- `required`
- `requiredMessage`
- `captionProperty` – при отображении экземпляра сущности задает имя атрибута, значение которого выводится в текстовом поле. Если данный атрибут не задан, компонент будет содержать `Instance Name` экземпляра.
- `metaClass` – если не установлен атрибут `datasource`, задает тип сущности, с которой будет работать компонент.
- `actions` – необязательный элемент, определяющий набор действий (кнопок справа)
  - `action` – элемент, определяющий действие. Кроме описания произвольного действия возможно использование стандартных действий, определяемых перечислением `PickerField.ActionType`: `lookup`, `clear`, `open`.

Если при объявлении компонента никаких действий не задано, загрузчик XML определит для него действия `lookup` и `clear`.

Пример использования компонента в **XML-дескрипторе** :

```
<pickerField id="pickerField1" datasource="carDs"
             property="model" width="250px">
```

```
<actions>
    <action id="clear"/>
    <action id="pickerAction1"
        caption=""
        icon="pickerfield/img/lookup-btn.png"
        invoke="someActionMethod1"/>
    <action id="pickerAction2"
        caption=""
        icon="pickerfield/img/lookup-btn.png"
        invoke="someActionMethod2"/>
</actions>
</pickerField>
```

## T

### Table

Служит для отображения данных в табличном виде.

Возможности:

- Выделение и множественное выделение строк в таблице.
- Сортировка по одной из колонок.
- Задание уникального стиля отображения для каждой ячейки.
- Применение форматирования для колонок.
- Использование для колонок функций агрегирования без модификации datasource.
- Возможность скрывать/показывать колонки таблицы.
- Возможность изменять порядок отображения колонок в таблице.
- Возможность самостоятельно задавать представление данных в ячейке.
- Возможность управлять настройками отображения по средством механизма представлений.

XML-имя компонента: **table** .

Атрибуты **table** :

- id
- visible
- styleName
- width

- height
- expandable
- editable
- presentations – необязательный атрибут, включение/выключение поддержки механизма представлений.
- sortable – необязательный атрибут, разрешает/запрещает сортировку в таблице. По-умолчанию имеет значение true.
- aggregatable – необязательный атрибут, разрешает/запрещает агрегацию в таблице. По-умолчанию имеет значение false.
- showTotalAggregation – необязательный атрибут, разрешает/запрещает отображение строки итоговой агрегации в таблице. По-умолчанию имеет значение true.
- multiselect – необязательный атрибут, разрешает/запрещает множественное выделение. По-умолчанию имеет значение false.
- actions – необязательный элемент, определяющий набор действий (кнопок справа)
  - action – элемент, определяющий действие
- buttonsPanel
- rowsCount – необязательный элемент, создающий для таблицы компонент `RowCount`, который позволяет загружать в таблицу данные постранично.
- columns – обязательный элемент, определяет набор колонок таблицы.
  - column – элемент, задающий опции для колонки таблицы. Может содержать следующие атрибуты:
    - id – обязательный атрибут, содержит название атрибута сущности, выводимого в колонке.
    - collapsed – необязательный атрибут, скрывает/показывает колонку. По-умолчанию

имеет значение `false`.

- `caption` – необязательный атрибут, содержит заголовок колонки.
- `width` – необязательный атрибут, отвечает за изначальную ширину колонки.
- `calculatable` – необязательный атрибут, используется только в редактируемой таблице. Означает, что значения в данной колонке являются вычислимыми и зависят от значений других колонок.
- `clickAction` – необязательный атрибут; содержит описание действия, которое будет выполнено при клике в ячейке.
- `editable` – необязательный атрибут, разрешает/запрещает редактирование данного атрибута в редактируемой таблице.
- `formatter`
- `aggregation`
- `validator`
- `rows` – обязательный элемент, декларирует используемый datasource, его атрибуты:
  - `datasource attribute`
  - `headerMode` – необязательный атрибут, вариант отображения заголовков рядов:
    - `NONE` - нет заголовков
    - `ICON` - пиктограмма

Пример использования компонента в XML-дескрипторе :

```
<table id="users" editable="false">
  <columns>
    <column id="login" caption="msg://login"
    clickAction="open:sec$User.edit"/>
    <column id="name" caption="msg://name"/>
    <column id="position" caption="msg://position"/>
```

```
<column id="group" caption="msg://group"/>
<column id="email" caption="msg://email"/>
<column id="active" caption="msg://active"/>
</columns>
<rows datasource="users"/>
</table>
```

## TabSheet

Контейнер, содержащий вкладки ( **tabs** ).

XML-имя компонента: **tabsheet** .

Атрибуты **tabsheet** :

- **id**
- **stylename**
- **height**
- **width**
- **visible**
- **expandable**

**tabsheet** должен содержать элементы **tab** , определяющие содержимое вкладок. Каждый элемент **tab** может содержать следующие атрибуты:

- **id** – идентификатор вкладки. Следует отметить, что вкладка не является компонентом, и данный идентификатор используется только в рамках **TabSheet** .
- **caption** – заголовок вкладки. Атрибут должен содержать ключ сообщения в *локализованном пакете сообщений* .
- **enable** . Это состояние вкладки. Может иметь значение **true** (доступна) или **false** (недоступна). По умолчанию **enable = "true"** .
- **lazy** . Задает отложенную загрузку содержимого вкладки. Может иметь значение **true** или **false** . При открытии экрана lazy-вкладки не загружают свое содержимого, что приводит к созданию меньшего количества компонентов в памяти и к меньшему количеству запросов к БД. Компоненты вкладки загружаются только в тот момент, когда пользователь выбирает данную вкладку. Если компоненты lazy-вкладки требуют инициализации, проводить ее

нужно не напрямую в методе `init()` *контроллера*, а в слушателе на переключение вкладок `TabSheet.TabChangeListener`.

**V**

## View

См. [Представление](#).

**X**

## XML-дескриптор

Описывает *источники данных* и расположение визуальных компонентов экрана.

XML-дескриптор содержит следующие элементы:

- ***window*** – корневой элемент. Атрибуты:
  1. *x m l n s*  
="http://schemas.haulmont.com/cuba/5.2/window.xsd" – схема XML
  2. *class* – имя класса *контроллера*
  3. *messagesPack* – имя *пакета сообщений*. Фактически это имя Java-пакета, содержащего файл *messages.properties*
  4. *caption* – заголовок экрана, должен содержать ссылку на сообщение из вышеуказанного пакета, например

```
caption="msg://caption"
```
  5. *focusComponent* – идентификатор компонента, который получит фокус ввода при открытии экрана (необязательно).
- ***metadataContext*** – опциональный элемент для инициализации *представлений* (views). Предпочтительным является определение всех представлений в одном общем файле *ext-views.xml* модуля core.
- ***dsContext*** – элемент, определяющий *источники данных*

данного экрана. Источники данных предназначены для реализации связанных с данными компонентов.

- *Layout* – элемент, определяющий компоновку экрана.