Создание приложений для СУБД Firebird с использованием различных компонент и драйверов:

Spring MVC и jOOQ

Автор: Денис Симонов
20.03.2017

Оглавление

Создание приложений с использованием jOOQ и Spring MVC

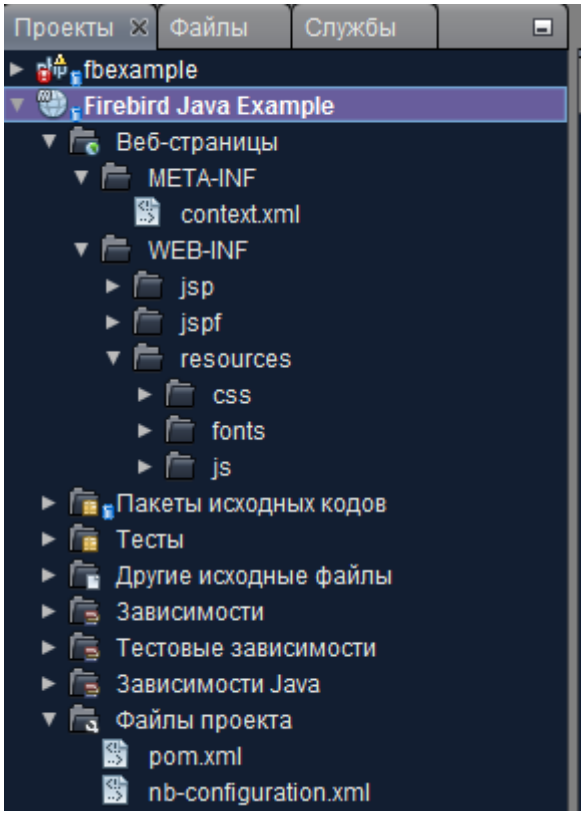
В данной статье будет описан процесс создания web приложения на языке Java с использованием фреймворка Spring MVC, библиотеки jOOQ и СУБД Firebird

Для упрощения разработки вы можете воспользоваться одной из распространённых IDE для Java (NetBeans, IntelliJ IDEA, Eclipse, JDeveloper или др.). Лично я использовал NetBeans. Для тестирования и отладки нам так же потребуется установить один и веб-серверов или серверов приложения (Apache Tomcat или Glass Fish) Создаём проект на основе шаблона Maven проекта веб-приложения.

После создания проекта на основе шаблона необходимо преобразовать его структуру папок так чтобы она была корректной для Spring 4. Если проект создавался в среде NetBeans 8.2, то необходимо выполнить следующие шаги:

1. Удалить файл index.html
2. Создать папку WEB-INF внутри папки Web Pages
3. Внутри папки WEB-INF создать папки jsp, jspf и resources
4. Внутри папки resources создаём папки js и CSS.
5. Внутри папки jsp создаём файл index.jsp.

После наших манипуляций структура папок должна выглядеть следующим образом.



В папке WEB-INF/jsp будут размещаться jsp странице, а в папке jspf части страниц, которые будут подключены в другие странице с помощью инструкции

<%@ include file=”<имя файла>” %>

Папка resource предназначена для размещения статических веб ресурсов. В папке WEB-INF/resources/css будут размещаться файлы каскадных таблиц стилей, в папке WEB-INF/resources/fonts – файлы шрифтов, в папке WEB-INF/resources/js – файлы JavaScript и сторонние JavaScript библиотеки.

Теперь поправим файл pom.xml и пропишем в него общие свойства приложения, зависимости от пакетов библиотек (Spring MVC, Jaybird, JDBC пул, JOOQ) и свойства JDBC подключения.

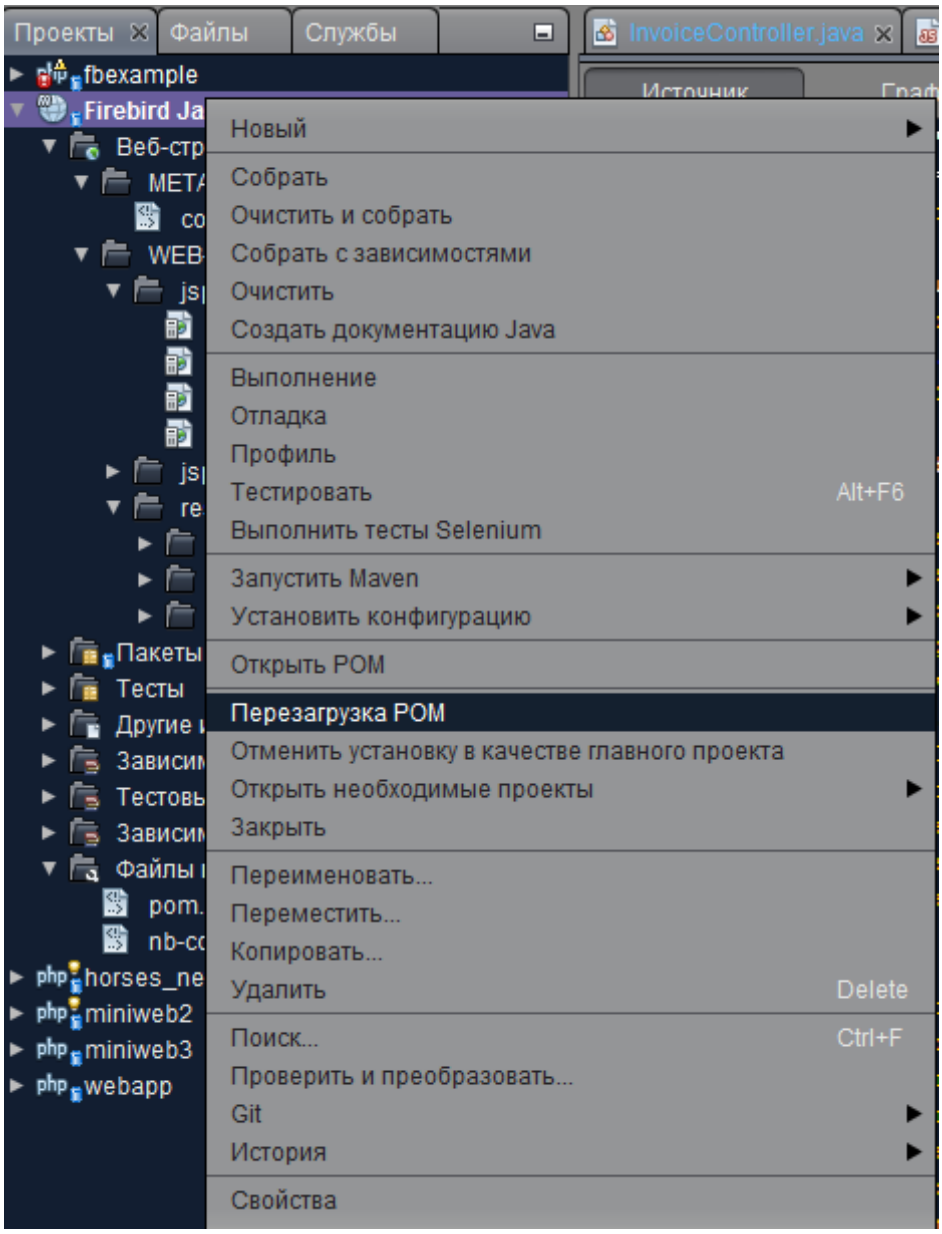
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>ru.ibase</groupId>
    <artifactId>fbjavaex</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <name>Firebird Java Example</name>

    <properties>
        <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <spring.version>4.3.4.RELEASE</spring.version>
        <jstl.version>1.2</jstl.version>
        <javax.servlet.version>3.0.1</javax.servlet.version>
        <db.url>jdbc:firebirdsql://localhost:3050/examples</db.url>
        <db.driver>org.firebirdsql.jdbc.FBDriver</db.driver>
        <db.username>SYSDBA</db.username>
        <db.password>masterkey</db.password>
    </properties>


    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-web-api</artifactId>
            <version>7.0</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>${javax.servlet.version}</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>${jstl.version}</version>
```

```
</dependency>

<!-- Работа с JSON -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.8.5</version>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.8.5</version>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.8.5</version>
</dependency>


<!-- Spring -->

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
</dependency>

<!-- JDBC -->

<dependency>
    <groupId>org.firebirdsql.jdbc</groupId>
    <artifactId>jaybird-jdk18</artifactId>
    <version>3.0.0</version>
</dependency>

<!-- Пул коннектов -->
```

```
    <dependency>
        <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>1.4</version>
    </dependency>

    <!-- jOOQ -->

    <dependency>
        <groupId>org.jooq</groupId>
        <artifactId>jooq</artifactId>
        <version>3.9.2</version>
    </dependency>

    <dependency>
        <groupId>org.jooq</groupId>
        <artifactId>jooq-meta</artifactId>
        <version>3.9.2</version>
    </dependency>

    <dependency>
        <groupId>org.jooq</groupId>
        <artifactId>jooq-codegen</artifactId>
        <version>3.9.2</version>
    </dependency>

    <!-- Testing -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <type>jar</type>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>${spring.version}</version>
        <scope>test</scope>
    </dependency>

</dependencies>

<build>

    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.1</version>
            <configuration>
                <source>1.7</source>
                <target>1.7</target>
                <compilerArguments>
                    <endorseddirs>${endorsed.dir}</endorseddirs>
                </compilerArguments>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
```

```
                <version>2.3</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <version>2.6</version>
                <executions>
                    <execution>
                        <phase>validate</phase>
                        <goals>
                            <goal>copy</goal>
                        </goals>
                        <configuration>
                            <outputDirectory>${endorsed.dir}</outputDirectory>
                            <silent>true</silent>
                            <artifactItems>
                                <artifactItem>
                                    <groupId>javax</groupId>
                                    <artifactId>javaee-endorsed-api</artifactId>
                                    <version>7.0</version>
                                    <type>jar</type>
                                </artifactItem>
                            </artifactItems>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

После того как вы прописали все необходимые зависимости, желательно перезагрузить POM, чтобы загрузить все необходимые библиотеки. Если этого не сделать, то в процессе работы с проектом могут возникать ошибки. В NetBeans это делается следующим образом



Мне не очень нравится конфигурирование через xml, поэтому я буду работать через классы конфигурации Java.

```
package ru.ibase.fbjavaex.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import org.springframework.web.servlet.view.JstlView;
import org.springframework.web.servlet.view.UrlBasedViewResolver;
import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
import org.springframework.http.converter.HttpMessageConverter;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import java.util.List;

@Configuration
@ComponentScan("ru.ibase.fbjavaex")
@EnableWebMvc
public class WebAppConfig extends WebMvcConfigurerAdapter {
```

```
    @Override
    public void configureMessageConverters(List<HttpMessageConverter<?>> httpMessageConverters) {
        MappingJackson2HttpMessageConverter jsonConverter = new
MappingJackson2HttpMessageConverter();
        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false);
        jsonConverter.setObjectMapper(objectMapper);
        httpMessageConverters.add(jsonConverter);
    }

    @Bean
    public UrlBasedViewResolver setupViewResolver() {
        UrlBasedViewResolver resolver = new UrlBasedViewResolver();
        resolver.setPrefix("/WEB-INF/jsp/");
        resolver.setSuffix(".jsp");
        resolver.setViewClass(JstlView.class);
        return resolver;
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**").addResourceLocations("/WEB-INF/resources/");
    }
}
```

В данном конфигурационном классе мы задаём место поиска веб ресурсов и JSP представлений. Метод `configureMessageConverters` устанавливает, что дата должна сериализоваться в строковое представление (по умолчанию сериализуется в числовом представлении как timestamp).

Теперь избавимся от файла Web.xml вместо него создадим файл WebInitializer.java.

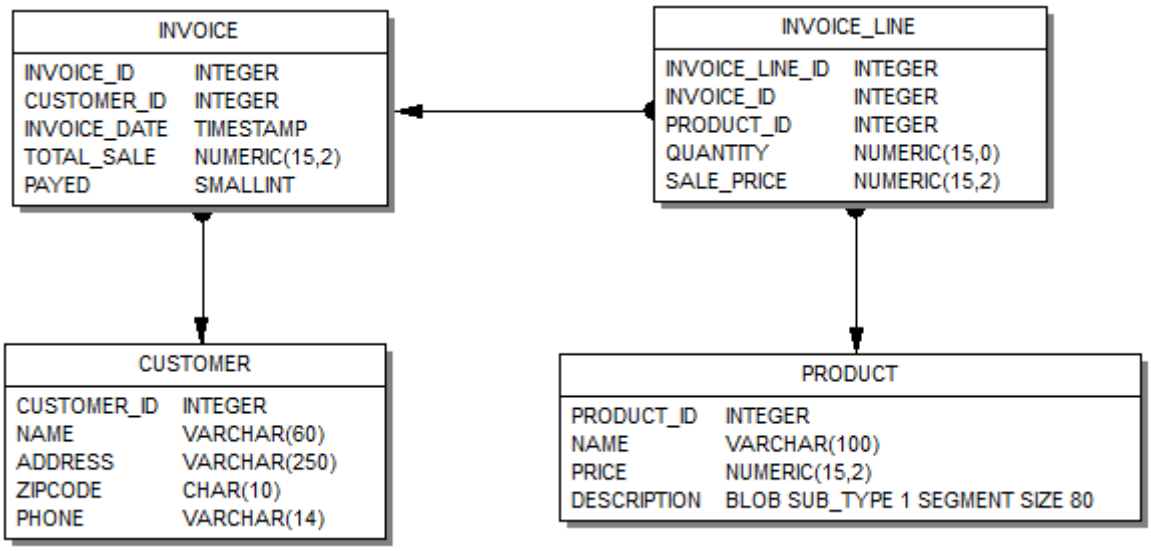
```
package ru.ibase.fbjavaex.config;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration.Dynamic;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class WebInitializer implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        AnnotationConfigWebApplicationContext ctx = new AnnotationConfigWebApplicationContext();
        ctx.register(WebAppConfig.class);
        ctx.setServletContext(servletContext);
        Dynamic servlet = servletContext.addServlet("dispatcher", new DispatcherServlet(ctx));
        servlet.addMapping("/");
        servlet.setLoadOnStartup(1);
    }

}
```

Осталось сконфигурировать IoC контейнеры для внедрения зависимостей. К этому шагу мы вернёмся позже, а сейчас перейдём к генерации классов для работы с базой данных через jOOQ.

Генерации классов для работы с базой данных через jOOQ

Работу с базой данных будем вести с помощью библиотеки jOOQ. jOOQ позволяет строить SQL запросы из объектов jOOQ и кода (наподобие LINQ). jOOQ имеет более тесную интеграцию с базой данных, чем ORM, поэтому кроме простых CRUD SQL запросов используемых в Active Record, позволяет использовать дополнительные возможности. Например, jOOQ умеет работать с хранимыми процедурами и функциями, последовательностями, использовать оконные функции и другие, специфичные для определённой СУБД, возможности. Полная документация по работе с jOOQ находится по адресу http://www.jooq.org/doc/3.9/manual-single-page/

Классы jOOQ для работы с базой данных генерируются на основе схемы базы данных. Наше приложение будет работать с базой данных, модель которой представлена на рисунке ниже.



Помимо таблиц, наша база данных содержит также хранимые процедуры и последовательности. В конце данной статьи приведена ссылка на скрипт создания базы данных.

Внимание!

Эта модель является просто примером. Ваша предметная область может быть сложнее, или полностью другой. Модель, используемая в этой статье, максимально упрощена для того, чтобы не загромождать описание работы с компонентами описанием создания и модификации модели данных.

Для генерации классов jOOQ, работающих с нашей БД, необходимо скачать следующие бинарные файлы по ссылке http://www.jooq.org/download или через maven репозиторий:

- **jooq-3.9.2.jar**

Главная библиотека, которая включается в наше приложение для работы с jOOQ.

- **jooq-meta-3.9.2.jar**
  Утилита, которая включается в вашу сборку для навигации по схеме базы данных через сгенерированные объекты.

- **jooq-codegen-3.9.2.jar**
  Утилита, которая включается в вашу сборку для генерации схемы базы данных.

Кроме того для подключения к БД Firebird через JDBC вам потребуется скачать драйвер [jaybird-full-3.0.0.jar].

Теперь надо создать файл конфигурации example.xml, который будет использован для генерации классов схемы БД.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration xmlns="http://www.jooq.org/xsd/jooq-codegen-3.8.0.xsd">
  <!-- Конфигурация подключения к БД -->
  <jdbc>
    <driver>org.firebirdsql.jdbc.FBDriver</driver>
    <url>jdbc:firebirdsql://localhost:3050/examples</url>
    <user>SYSDBA</user>
    <password>masterkey</password>
    <properties>
      <property>
        <key>charSet</key>
        <value>utf-8</value>
      </property>
    </properties>
  </jdbc>

  <generator>
    <name>org.jooq.util.JavaGenerator</name>

    <database>
      <!-- Тип базы данных. Формат:
           org.util.[database].[database]Database -->
      <name>org.jooq.util.firebird.FirebirdDatabase</name>

      <inputSchema></inputSchema>

      <!-- Все объекты, которые генерируются из вашей схемы
          (Регулярное выражение Java.
           Используйте фильтры, чтобы ограничить количество объектов).
           Следите за чувствительностью к регистру. В зависимости от вашей базы данных,
           это может быть важно! -->
      <includes>.*</includes>

      <!-- Объекты, которые исключаются при генерации из вашей схемы.
           (Регулярное выражение Java).
           В данном случае мы исключаем системные таблицы RDB$, таблицы мониторинга MON$
           и псевдотаблицы безопасности SEC$. -->
      <excludes>
          RDB\$.*
        | MON\$.*
        | SEC\$.*
      </excludes>
    </database>
```

```
    <target>
      <!-- Имя пакета в который будут выгружены сгенерированные классы -->
      <packageName>ru.ibase.fbjavaex.exampledb</packageName>

      <!-- Директория для размещения сгенерированных классов.
           Здесь используется структура директорий Maven. -->
      <directory>e:/OpenServer/domains/localhost/fbjavaex/src/main/java/</directory>
    </target>
  </generator>
</configuration>
```

Теперь переходим в командную строку и выполняем следующую команду:

```
java -cp jooq-3.9.2.jar;jooq-meta-3.9.2.jar;jooq-codegen-3.9.2.jar;jaybird-full-3.0.0.jar;.
org.jooq.util.GenerationTool example.xml
```

Данная команда создаст необходимые классы и позволит писать на языке Java запросы к объектам БД. Подробнее с процессом генерации классов вы можете ознакомиться по ссылке https://www.jooq.org/doc/3.9/manual-single-page/#code-generation.

Конфигурация IoC контейнеров

В Spring внедрение зависимостей (Dependency Injection (DI)) осуществляется через Spring IoC (Inversion of Control) контейнер. Внедрение зависимостей, является процессом, согласно которому объекты определяют свои зависимости, т.е. объекты, с которыми они работают, через аргументы конструктора/фабричного метода или свойства, которые были установлены или возвращены фабричным методом. Затем контейнер inject(далее "внедряет") эти зависимости при создании бина. Подробнее о внедрении зависимостей вы можете почитать по ссылке http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans.

Я не сторонник xml конфигурации, поэтому мы будем использовать подход на основе аннотаций и Java-конфигурации. Основными признаками и частями Java-конфигурации IoC контейнера являются классы с аннотацией @Configuration и методы с аннотацией @Bean. Аннотация @Bean используется для указания того, что метод создает, настраивает и инициализирует новый объект, управляемый Spring IoC контейнером. Такие методы можно использовать как в классах с аннотацией @Configuration. Наш IoC контейнер будет возвращать пул подключений, менеджер транзакций, транслятор исключений (преобразует исключения SQLException в специфичные для Spring исключения DataAccessException), DSL контекст (стартовая точка, для построения всех запросов используя Fluent API), а также менеджеры для реализации бизнес логики и гриды для отображения данных.

```
/**
 * Конфигурация IoC контейнера
```

```
 * для осуществления внедрения зависимостей.
 */

package ru.ibase.fbjavaex.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.sql.DataSource;
import org.apache.commons.dbcp.BasicDataSource;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy;
import org.jooq.impl.DataSourceConnectionProvider;
import org.jooq.DSLContext;
import org.jooq.impl.DefaultDSLContext;
import org.jooq.impl.DefaultConfiguration;
import org.jooq.SQLDialect;
import org.jooq.impl.DefaultExecuteListenerProvider;

import ru.ibase.fbjavaex.exception.ExceptionTranslator;

import ru.ibase.fbjavaex.managers.*;
import ru.ibase.fbjavaex.jqgrid.*;

/**
 * Конфигурационный класс Spring IoC контейнера
 */
@Configuration
public class JooqConfig {


    /**
     * Возвращает пул коннектов
     *
     * @return
     */
    @Bean(name = "dataSource")
    public DataSource getDataSource() {
        BasicDataSource dataSource = new BasicDataSource();
        // определяем конфигурацию подключения
        dataSource.setUrl("jdbc:firebirdsql://localhost:3050/examples");
        dataSource.setDriverClassName("org.firebirdsql.jdbc.FBDriver");
        dataSource.setUsername("SYSDBA");
        dataSource.setPassword("masterkey");
        dataSource.setConnectionProperties("charSet=utf-8");
        return dataSource;
    }

    /**
     * Возращает менеджер транзакций
     *
     * @return
     */
    @Bean(name = "transactionManager")
    public DataSourceTransactionManager getTransactionManager() {
        return new DataSourceTransactionManager(getDataSource());
    }

    @Bean(name = "transactionAwareDataSource")
    public TransactionAwareDataSourceProxy getTransactionAwareDataSource() {
        return new TransactionAwareDataSourceProxy(getDataSource());
    }

    /**
     * Возвращает провайдер подключений
     *
     * @return
     */
    @Bean(name = "connectionProvider")
```

```
    public DataSourceConnectionProvider getConnectionProvider() {
        return new DataSourceConnectionProvider(getTransactionAwareDataSource());
    }

    /**
     * Возвращает транслятор исключений
     *
     * @return
     */
    @Bean(name = "exceptionTranslator")
    public ExceptionTranslator getExceptionTranslator() {
        return new ExceptionTranslator();
    }

    /**
     * Возвращает конфигурацию DSL контекста
     *
     * @return
     */
    @Bean(name = "dslConfig")
    public org.jooq.Configuration getDslConfig() {
        DefaultConfiguration config = new DefaultConfiguration();
        // используем диалект SQL СУБД Firebird
        config.setSQLDialect(SQLDialect.FIREBIRD);
        config.setConnectionProvider(getConnectionProvider());
        DefaultExecuteListenerProvider listenerProvider = new
DefaultExecuteListenerProvider(getExceptionTranslator());
        config.setExecuteListenerProvider(listenerProvider);
        return config;
    }

    /**
     * Возвращает DSL контекст
     *
     * @return
     */
    @Bean(name = "dsl")
    public DSLContext getDsl() {
        org.jooq.Configuration config = this.getDslConfig();
        return new DefaultDSLContext(config);
    }

    /**
     * Возвращает менеджер заказчиков
     *
     * @return
     */
    @Bean(name = "customerManager")
    public CustomerManager getCustomerManager() {
        return new CustomerManager();
    }

    /**
     * Возвращает грид с заказчиками
     *
     * @return
     */
    @Bean(name = "customerGrid")
    public JqGridCustomer getCustomerGrid() {
        return new JqGridCustomer();
    }

    /**
     * Возвращает менеджер продуктов
     *
     * @return
     */
```

```
    @Bean(name = "productManager")
    public ProductManager getProductManager() {
        return new ProductManager();
    }

    /**
     * Возвращает грид с товарами
     *
     * @return
     */
    @Bean(name = "productGrid")
    public JqGridProduct getProductGrid() {
        return new JqGridProduct();
    }

    /**
     * Возвращает менеджер счёт фактур
     *
     * @return
     */
    @Bean(name = "invoiceManager")
    public InvoiceManager getInvoiceManager() {
        return new InvoiceManager();
    }

    /**
     * Возвращает грид с заголовками счёт фактур
     *
     * @return
     */
    @Bean(name = "invoiceGrid")
    public JqGridInvoice getInvoiceGrid() {
        return new JqGridInvoice();
    }

    /**
     * Возвращает грид с позициями счёт фактуры
     *
     * @return
     */
    @Bean(name = "invoiceLineGrid")
    public JqGridInvoiceLine getInvoiceLineGrid() {
        return new JqGridInvoiceLine();
    }

    /**
     * Возвращает рабочий период
     *
     * @return
     */
    @Bean(name = "workingPeriod")
    public WorkingPeriod getWorkingPeriod() {
        return new WorkingPeriod();
    }

}
```

Построение SQL запросов используя jOOQ

Прежде чем рассматривать реализацию менеджеров и сеток (grids) расскажем, как работать с базой данных через jOOQ. Здесь будут изложены лишь краткие сведения о построении запросов, полную документацию по этому вопросу вы можете найти в главе [sql-building](#) документации jOOQ.

Класс org.jooq.impl.DSL является основным классом, от которого вы будете создавать все объекты jOOQ. Он выступает в роли статической фабрики для табличных выражений, выражений столбцов (или полей), условных выражений и многих других частей запроса.

DSLContext ссылается на объект org.jooq.Configuration, который настраивает поведение jOOQ, при выполнении запросов. В отличие от статического DSL, DSLContext позволяет создавать SQL-операторы, которые уже "настроены" и готовы к выполнению. В нашем приложении DSLContext создаётся в классе конфигурации JooqConfig в методе getDsl. Конфигурация для DSLContext возвращается методом getDslConfig. В этом методе мы указали, что будем использовать диалект SQL СУБД Firebird, провайдер подключений (определяет, как мы получаем подключение через JDBC) и слушатель выполнения SQL запросов.

jOOQ поставляется с собственным DSL (или Domain Specific Language), который эмулирует SQL в Java. Это означает, что вы можете писать SQL-операторы почти так, как если бы Java изначально поддерживал их, примерно так же, как .NET в C# делает это с помощью LINQ к SQL.

jOOQ использует неформальную BNF нотацию, которая моделирует унифицированный SQL диалект, подходящий для большинства СУБД. В отличие от других, более простых фреймворков, которые используют "Fluent API" или "метод цепочек", иерархия интерфейса BNF на основе jOOQ не позволяет плохой синтаксис запросов.

Давайте рассмотрим простой запрос на языке SQL

```
SELECT *
  FROM author a
  JOIN book b ON a.id = b.author_id
 WHERE a.year_of_birth > 1920
   AND a.first_name = 'Paulo'
 ORDER BY b.title
```

В jOOQ он будет выглядеть следующим образом:

```
Result<Record> result =
dsl.select()
   .from(AUTHOR.as("a"))
   .join(BOOK.as("b")).on(a.ID.equal(b.AUTHOR_ID))
   .where(a.YEAR_OF_BIRTH.greaterThan(1920)
   .and(a.FIRST_NAME.equal("Paulo")))
   .orderBy(b.TITLE)
   .fetch();
```

Классы AUTHOR и BOOK, описывающие соответствующие таблицы должны быть сгенерированы заранее. Процесс генерации классов jOOQ по заданной схеме БД был описан выше.

В данном случае мы задали таблицам AUTHOR и BOOK алиас с помощью конструкции as. Без использования алиасов этот запрос выглядел бы следующим образом

```
Result<Record> result =
dsl.select()
   .from(AUTHOR)
   .join(BOOK).on(AUTHOR.ID.equal(BOOK.AUTHOR_ID))
   .where(AUTHOR.YEAR_OF_BIRTH.greaterThan(1920)
   .and(AUTHOR.FIRST_NAME.equal("Paulo")))
   .orderBy(BOOK.TITLE)
   .fetch();
```

Теперь посмотрим более сложный запрос с использованием агрегатных функций и группировки.

```
SELECT AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME, COUNT(*)
FROM AUTHOR
  JOIN BOOK ON AUTHOR.ID = BOOK.AUTHOR_ID
WHERE BOOK.LANGUAGE = 'DE'
  AND BOOK.PUBLISHED > '2008-01-01'
GROUP BY AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME
  HAVING COUNT(*) > 5
ORDER BY AUTHOR.LAST_NAME ASC NULLS FIRST
  OFFSET 1 ROWS
  FETCH FIRST 2 ROWS ONLY
```

В jOOQ он будет выглядеть так:

```
dsl.select(AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME, count())
   .from(AUTHOR)
   .join(BOOK).on(BOOK.AUTHOR_ID.equal(AUTHOR.ID))
   .where(BOOK.LANGUAGE.equal("DE"))
   .and(BOOK.PUBLISHED.greaterThan("2008-01-01"))
   .groupBy(AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME)
   .having(count().greaterThan(5))
   .orderBy(AUTHOR.LAST_NAME.asc().nullsFirst())
   .limit(2)
   .offset(1)
   .fetch();
```

Заметьте ограничение на количество возвращаемых записей, будет сгенерировано в соответствии с указанным диалектом SQL. В примере выше использовался диалект FIREIRD_3_0. Если бы был указан диалект FIREBIRD_2_5 или просто FIREBIRD, то использовалось бы предложение ROWS вместо OFSET … FETCH.

Вы можете собирать запрос по частям. Это позволяет менять его динамически, что можно использовать для изменения порядка сортировки или добавления дополнительных параметров фильтрации.

```
SelectFinalStep<?> select
   = dsl.select()
        .from(PRODUCT);

SelectQuery<?> query = select.getQuery();
switch (searchOper) {
    case "eq":
        query.addConditions(PRODUCT.NAME.eq(searchString));
        break;
    case "bw":
```

```
                query.addConditions(PRODUCT.NAME.startsWith(searchString));
                break;
            case "cn":
                query.addConditions(PRODUCT.NAME.contains(searchString));
                break;
        }
        switch (sOrd) {
            case "asc":
                query.addOrderBy(PRODUCT.NAME.asc());
                break;
            case "desc":
                query.addOrderBy(PRODUCT.NAME.desc());
                break;
        }
        return query.fetchMaps();
```

Именованные и неименованные параметры

По умолчанию каждый раз, когда вы используете в запросе литера строк, дат и чисел, а также подставляете внешние переменные, jOOQ делает привязку этой переменной или литерала через неименованные параметры. Например, следующее выражение на языке Java

```
dsl.select()
   .from(BOOK)
   .where(BOOK.ID.equal(5))
   .and(BOOK.TITLE.equal("Animal Farm"))
   .fetch();
```

Эквивалентно полной форме записи

```
dsl.select()
   .from(BOOK)
   .where(BOOK.ID.equal(val(5)))
   .and(BOOK.TITLE.equal(val("Animal Farm")))
   .fetch();
```

и преобразуется в sql запрос

```
SELECT *
FROM BOOK
WHERE BOOK.ID = ?
  AND BOOK.TITLE = ?
```

Вам не нужно беспокоиться какой индекс у соответствующего параметра, значения автоматически будут привязаны к нужному параметру. Если нужно изменить значение параметра, то вы можете сделать это, выбрав нужный параметр по номеру индекса (индексация начинается с 1).

```
Select<?> select =
  dsl.select()
     .from(BOOK)
     .where(BOOK.ID.equal(5))
     .and(BOOK.TITLE.equal("Animal Farm"));
Param<?> param = select.getParam("2");
Param.setValue("Animals as Leaders");
```

Другим способом присвоить параметру новое значение является вызов метода bind.

```
Query query1 =
  dsl.select()
     .from(AUTHOR)
     .where(LAST_NAME.equal("Poe"));
query1.bind(1, "Orwell");
```

Кроме того, jOOQ поддерживает именованные параметры. В этом случае их надо явно создавать, используя org.jooq.Param.

```
// Create a query with a named parameter. You can then use that name for
// accessing the parameter again
Query query1 =
  dsl.select()
     .from(AUTHOR)
     .where(LAST_NAME.equal(param("lastName", "Poe")));
Param<?> param1 = query.getParam("lastName");

// Or, keep a reference to the typed parameter in order not to lose the <T> type information:
Param<String> param2 = param("lastName", "Poe");
Query query2 =
  dsl.select()
     .from(AUTHOR)
     .where(LAST_NAME.equal(param2));

// You can now change the bind value directly on the Param reference:
param2.setValue("Orwell");
```

Другим способом присвоить параметру новое значение является вызов метода bind.

```
// Or, with named parameters
Query query2 =
  dsl.select()
     .from(AUTHOR)
     .where(LAST_NAME.equal(param("lastName", "Poe")));
query2.bind("lastName", "Orwell");
```

Возврат значений из селективных запросов

jOOQ предоставляет множество способов извлечения данных из SQL запросов. Мы не будем перечислять здесь все способы, подробнее вы можете прочитать о них в главе Fetching документации jOOQ. Мы в своём примере будем пользоваться возвратом в список карт (метод fetchMaps), который удобно использовать для сериализации результата в JSON.

Другие типы запросов

Теперь посмотрим, как выглядят другие типы запросов. Например, следующий запрос для вставки записи

```
INSERT INTO AUTHOR
       (ID, FIRST_NAME, LAST_NAME)
```

```
VALUES (100, 'Hermann', 'Hesse');
```

в jOOQ будет выглядеть так

```
dsl.insertInto(AUTHOR,
        AUTHOR.ID, AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME)
   .values(100, "Hermann", "Hesse")
   .execute();
```

Вот такой запрос для обновления записи

```
UPDATE AUTHOR
   SET FIRST_NAME = 'Hermann',
       LAST_NAME = 'Hesse'
 WHERE ID = 3;
```

с использованием jOOQ записывается следующим образом

```
dsl.update(AUTHOR)
   .set(AUTHOR.FIRST_NAME, "Hermann")
   .set(AUTHOR.LAST_NAME, "Hesse")
   .where(AUTHOR.ID.equal(3))
   .execute();
```

Запрос на удаление записи

```
DELETE FROM AUTHOR
 WHERE ID = 100;
```

Выглядит так

```
dsl.delete(AUTHOR)
   .where(AUTHOR.ID.equal(100))
   .execute();
```

Кроме того, в jOOQ можете строить более сложные модифицирующие запросы, например MERGE.

Большим преимуществом jOOQ является поддержка работы с хранимыми процедурами. Хранимые процедуры извлекаются в пакет *.Routines.* после чего с ними можно удобно работать, например следующий код на Java

```
        int invoiceId = dsl.nextval(GEN_INVOICE_ID).intValue();

        spAddInvoice(dsl.configuration(),
                invoiceId,
                customerId,
                invoiceDate);
```

Эквивалентен получению следующего значения генератора с помощью SQL запроса

```
SELECT NEXT VALUE FOR GEN_INVOICE_ID FROM RDB$DATABASE
```

И последующего вызова хранимой процедуры

```
EXECUTE PROCEDURE SP_ADD_INVOICE(:INVOICE_ID, :CUSTOMER_ID, :INVOICE_DATE);
```

jOOQ также предоставляет вам средства для построения простых DDL запросов, но мы не будет их рассматривать здесь.

Работа с транзакциями

По умолчанию jOOQ работает в режиме авто подтверждения транзакции, т.е. на каждый SQL оператор стартует новая транзакция, и если в процессе выполнения SQL оператора не было ошибок транзакция подтверждается, в противном случае откатывается. По умолчанию используется транзакция с параметрами READ_WRITE READ_COMMITED REC_VERSION WAIT. То есть те же самые что используются JDBC драйвером. Изменить режим изолированности по умолчанию можно через параметры пула соединений (см. BasicDataSource.setDefaultTransactionIsolation в методе getDataSource класса конфигурации JooqConfig).

В jOOQ существует несколько способов явного управления транзакциями. Поскольку мы разрабатываем приложение с использованием Spring Framework, будем использовать менеджер транзакций заданный в конфигурации (JooqConfig). Получить менеджер транзакций можно, объявив в классе свойство txMgr следующим образом:

```
…
    @Autowired
    private DataSourceTransactionManager txMgr;
…
```

В этом случае типичный сценарий работы с транзакцией выглядит следующим образом:

```
TransactionStatus tx = txMgr.getTransaction(new DefaultTransactionDefinition());
try {
    // действия внутри транзакции
    for (int i = 0; i < 2; i++)
        dsl.insertInto(BOOK)
           .set(BOOK.ID, 5)
           .set(BOOK.AUTHOR_ID, 1)
           .set(BOOK.TITLE, "Book 5")
           .execute();
    // подтверждение транзакции
    txMgr.commit(tx);
}
catch (DataAccessException e) {
    // откат транзакции
    txMgr.rolback(tx);
}
```

Однако Spring позволяет осуществить подобный сценарий намного проще с помощью аннотации @Transactional указанной перед методом класса. В этом случае все действия, производимые в методе, будут обёрнуты транзакцией.

```java
/**
 * Удаление заказчика
 *
 * @param customerId
 */
@Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
public void delete(int customerId) {
    this.dsl.deleteFrom(CUSTOMER)
            .where(CUSTOMER.CUSTOMER_ID.eq(customerId))
            .execute();
}
```

Параметр propagation задаёт, каким образом будет вестись работа с транзакциями, если наш метод вызывается из внешней транзакции.

- *Propagation.REQUIRED* — выполняться в существующей транзакции, если она есть, иначе создавать новую.
- *Propagation.MANDATORY* — выполняться в существующей транзакции, если она есть, иначе генерировать исключение.
- *Propagation.SUPPORTS* — выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции.
- *Propagation.NOT_SUPPORTED* — всегда выполняться вне транзакции. Если есть существующая, то она будет остановлена.
- *Propagation.REQUIRES_NEW* — всегда выполняться в новой независимой транзакции. Если есть существующая, то она будет остановлена до окончания выполнения новой транзакции.
- *Propagation.NESTED* — если есть текущая транзакция, выполняться в новой, так называемой, вложенной транзакции. Если вложенная транзакция будет отменена, то это не повлияет на внешнюю транзакцию; если будет отменена внешняя транзакция, то будет отменена и вложенная. Если текущей транзакции нет, то просто создаётся новая.
- *Propagation.NEVER* — всегда выполнять вне транзакции, при наличии существующей генерировать исключение.

Параметр isolation указывает режим изолированности транзакции. Поддерживается 5 значений: DEFAULT, READ_UNCOMMITTED, READ_COMMITTED, REPEATABLE_READ, SERIALIZABLE. Если указано значение параметра DEFAULT, то будет использоваться умолчательный режим изолированности транзакции. Остальные режимы изолированности взяты из SQL стандарта. В Firebird несколько другие режимы изолированности и полностью соответствует всем критериям лишь режим READ_COMMITED. Таким образом, режим в JDBC READ_COMMITTED отображает на read_committed в Firebird, REPEATABLE_READ – concurrency (Snapshot), а SERIALIZABLE – consistency. Кроме того, помимо режима изолированности, Firebird поддерживает дополнительные параметры транзакции (NO RECORD_VERSION и RECORD_VERSION, WAIT и NO WAIT). Вы можете настроить отображение стандартных уровней изолированности на параметры транзакции Firebird с помощью задания свойств JDBC соединения (подробнее см. в Jaybird 2.1 JDBC driver Java Programmer's Manual в главе Using transactions). Если ваша транзакция работает более чем с 1 запросом, то рекомендуется режим изолированности REPEATABLE_READ для обеспечения согласованности данных.

В аннотации @Transactional вы можете задать, является ли транзакция только для чтения с помощью свойства readOnly. По умолчанию транзакция находится в режиме read-write.

Написание кода приложения

Данные нашего приложения мы будем отображать с помощью JavaScript компонента jqGrid. В настоящий момент jqGrid распространяется по коммерческой лицензии, но бесплатен для некоммерческих целей. Вместо него вы можете воспользоваться форком free-jqGrid. Для отображения данных в данном гриде и элементов постраничной навигации нам требуется вернуть данные в формате JSON, структура которых выглядит следующим образом.

```
{
    total: 100,
    page: 3,
    records: 3000,
    rows: [
       {id: 1, name: “Ada”},
       {id: 2, name: “Smith”},
      …
    ]
}
```

где

- total – общее количество страниц;
- page – номер текущей страницы;
- records – общее количество записей;
- rows – массив записей на текущей странице.

Создадим класс который описывает данную струтуру:

```
package ru.ibase.fbjavaex.jqgrid;

import java.util.List;
import java.util.Map;

/**
 * Класс описывающий структуру которая используется jqGrid
 * Предназначен для сериализации в JSON
 *
 * @author Simonov Denis
 */
public class JqGridData {

    /**
     * Total number of pages
     */
    private final int total;

    /**
     * The current page number
     */
    private final int page;
```

```
    /**
     * Total number of records
     */
    private final int records;

    /**
     * The actual data
     */
    private final List<Map<String, Object>> rows;

    /**
     * Конструктор
     *
     * @param total
     * @param page
     * @param records
     * @param rows
     */
    public JqGridData(int total, int page, int records, List<Map<String, Object>> rows) {
        this.total = total;
        this.page = page;
        this.records = records;
        this.rows = rows;
    }

    /**
     * Возвращает общее количество страниц
     *
     * @return
     */
    public int getTotal() {
        return total;
    }

    /**
     * Возвращает текущую страницу
     *
     * @return
     */
    public int getPage() {
        return page;
    }

    /**
     * Возвращает общее количество записей
     *
     * @return
     */
    public int getRecords() {
        return records;
    }

    /**
     * Возвращает список карт
     * Это массив данных для отображения в гриде
     *
     * @return
     */
    public List<Map<String, Object>> getRows() {
        return rows;
    }
}
```

Теперь напишем абстрактный класс, который будет возвращать вышеописанную структуру в зависимости от условий поиска и сортировки. Этот класс будет предком классов возвращающие подобные структуры для конкретных сущностей.

```
/*
 * Абстрактный класс для работы с JqGrid
 */
package ru.ibase.fbjavaex.jqgrid;

import java.util.Map;
import java.util.List;
import org.jooq.DSLContext;
import org.springframework.beans.factory.annotation.Autowired;

/**
 * Работа с JqGrid
 *
 * @author Simonov Denis
 */
public abstract class JqGrid {


    @Autowired(required = true)
    protected DSLContext dsl;

    protected String searchField = "";
    protected String searchString = "";
    protected String searchOper = "eq";
    protected Boolean searchFlag = false;
    protected int pageNo = 0;
    protected int limit = 0;
    protected int offset = 0;
    protected String sIdx = "";
    protected String sOrd = "asc";

    /**
     * Возвращает общее количество записей
     *
     * @return
     */
    public abstract int getCountRecord();

    /**
     * Возвращает структуру для сериализации в JSON
     *
     * @return
     */
    public JqGridData getJqGridData() {
        int recordCount = this.getCountRecord();
        List<Map<String, Object>> records = this.getRecords();

        int total = 0;
        if (this.limit > 0) {
            total = recordCount / this.limit + 1;
        }

        JqGridData jqGridData = new JqGridData(total, this.pageNo, recordCount, records);
        return jqGridData;
    }


    /**
     * Возвращает количество записей на странице
     *
     * @return
     */
    public int getLimit() {
```

```
    return this.limit;
}

/**
 * Возвращает смещение до извлечения первой записи
 *
 * @return
 */
public int getOffset() {
    return this.offset;
}

/**
 * Возвращает имя поля для сортировки
 *
 * @return
 */
public String getIdx() {
    return this.sIdx;
}

/**
 * Возвращает порядок сортировки
 *
 * @return
 */
public String getOrd() {
    return this.sOrd;
}

/**
 * Возвращает номер текущей страницы
 *
 * @return
 */
public int getPageNo() {
    return this.pageNo;
}

/**
 * Возвращает массив записей как список карт
 *
 * @return
 */
public abstract List<Map<String, Object>> getRecords();

/**
 * Возвращает поле для поиска
 *
 * @return
 */
public String getSearchField() {
    return this.searchField;
}

/**
 * Возвращает значение для поиска
 *
 * @return
 */
public String getSearchString() {
    return this.searchString;
}

/**
 * Возвращает операцию поиска
```

```
     *
     * @return
     */
    public String getSearchOper() {
        return this.searchOper;
    }

    /**
     * Устанавливает ограничение на количество выводимых записей
     *
     * @param limit
     */
    public void setLimit(int limit) {
        this.limit = limit;
    }

    /**
     * Устанавливает количество записей, которые надо пропустить
     *
     * @param offset
     */
    public void setOffset(int offset) {
        this.offset = offset;
    }

    /**
     * Устанавливает сортировку
     *
     * @param sIdx
     * @param sOrd
     */
    public void setOrderBy(String sIdx, String sOrd) {
        this.sIdx = sIdx;
        this.sOrd = sOrd;
    }

    /**
     * Устанавливает номер текущей страницы
     *
     * @param pageNo
     */
    public void setPageNo(int pageNo) {
        this.pageNo = pageNo;
        this.offset = (pageNo - 1) * this.limit;
    }

    /**
     * Устанавливает условие поиска
     *
     * @param searchField
     * @param searchString
     * @param searchOper
     */
    public void setSearchCondition(String searchField, String searchString, String searchOper) {
        this.searchFlag = true;
        this.searchField = searchField;
        this.searchString = searchString;
        this.searchOper = searchOper;
    }
}
```

Обратите внимание, данный класс содержит свойство `DSLContext dsl`, которое будет использоваться для построения запросов на выборку данных с помощью jOOQ.

Создание справочников

Теперь мы можем приступить к созданию справочников. Мы опишем процесс создания справочников на примере справочника заказчиков. Справочник продуктов создаётся схожим образом, и при желании вы можете просмотреть его исходный код по ссылке, приведённой в конце статьи.

Сначала реализуем класс для работы с jqGrid, он будет наследоваться от нашего абстрактного класса `ru.ibase.fbjavaex.jqgrid.JqGrid` описанного выше. В нём имеется возможность поиска и разнонаправленной сортировки по полю NAME. В листинге исходного кода будут приведены поясняющие комментарии.

```
package ru.ibase.fbjavaex.jqgrid;

import org.jooq.*;
import java.util.List;
import java.util.Map;

import static ru.ibase.fbjavaex.exampledb.Tables.CUSTOMER;

/**
 * Грид заказчиков
 *
 * @author Simonov Denis
 */
public class JqGridCustomer extends JqGrid {

    /**
     * Добавление условия поиска
     *
     * @param query
     */
    private void makeSearchCondition(SelectQuery<?> query) {
        switch (this.searchOper) {
            case "eq":
                // CUSTOMER.NAME = ?
                query.addConditions(CUSTOMER.NAME.eq(this.searchString));
                break;
            case "bw":
                // CUSTOMER.NAME STARTING WITH ?
                query.addConditions(CUSTOMER.NAME.startsWith(this.searchString));
                break;
            case "cn":
                // CUSTOMER.NAME CONTAINING ?
                query.addConditions(CUSTOMER.NAME.contains(this.searchString));
                break;
        }
    }


    /**
     * Возвращает общее количество записей
     *
     * @return
     */
    @Override
    public int getCountRecord() {
        // запрос, возвращающий количество записей
        SelectFinalStep<?> select
            = dsl.selectCount()
```

```
                .from(CUSTOMER);

        SelectQuery<?> query = select.getQuery();
        // если мы осуществляем поиск, то добавляем условие поиска
        if (this.searchFlag) {
            makeSearchCondition(query);
        }
        // возарщаем количество
        return (int) query.fetch().getValue(0, 0);
    }

    /**
     * Возвращает записи грида
     *
     * @return
     */
    @Override
    public List<Map<String, Object>> getRecords() {
        // Базовый запрос на выборку
        SelectFinalStep<?> select =
            dsl.select()
               .from(CUSTOMER);

        SelectQuery<?> query = select.getQuery();
        // если мы осуществляем поиск, то добавляем условие поиска
        if (this.searchFlag) {
            makeSearchCondition(query);
        }
        // задаём порядок сортировки
        switch (this.sOrd) {
            case "asc":
                query.addOrderBy(CUSTOMER.NAME.asc());
                break;
            case "desc":
                query.addOrderBy(CUSTOMER.NAME.desc());
                break;
        }
        // ограничиваем количество записей
        if (this.limit != 0) {
            query.addLimit(this.limit);
        }
        // смещение
        if (this.offset != 0) {
            query.addOffset(this.offset);
        }
        // возвращаем массив карт
        return query.fetchMaps();
    }
}
```

Добавление, редактирование и удаление заказчика мы будем осуществлять через класс CustomerManager, который является своеобразным бизнес слоем между соответствующим контроллером и базой данных. Все операции в этом слое мы будем осуществлять в транзакции с уровнем изолированности Snapshot.

```
package ru.ibase.fbjavaex.managers;

import org.jooq.DSLContext;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Isolation;

import static ru.ibase.fbjavaex.exampledb.Tables.CUSTOMER;
```

```java
import static ru.ibase.fbjavaex.exampledb.Sequences.GEN_CUSTOMER_ID;

/**
 * Менеджер заказчиков
 *
 * @author Simonov Denis
 */
public class CustomerManager {

    @Autowired(required = true)
    private DSLContext dsl;

    /**
     * Добавление заказчика
     *
     * @param name
     * @param address
     * @param zipcode
     * @param phone
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
    public void create(String name, String address, String zipcode, String phone) {
        if (zipcode != null) {
            if (zipcode.trim().isEmpty()) {
                zipcode = null;
            }
        }

        int customerId = this.dsl.nextval(GEN_CUSTOMER_ID).intValue();

        this.dsl
                .insertInto(CUSTOMER,
                        CUSTOMER.CUSTOMER_ID,
                        CUSTOMER.NAME,
                        CUSTOMER.ADDRESS,
                        CUSTOMER.ZIPCODE,
                        CUSTOMER.PHONE)
                .values(
                        customerId,
                        name,
                        address,
                        zipcode,
                        phone
                )
                .execute();
    }

    /**
     * Редактирование заказчика
     *
     * @param customerId
     * @param name
     * @param address
     * @param zipcode
     * @param phone
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
    public void edit(int customerId, String name, String address, String zipcode, String phone) {

        if (zipcode != null) {
            if (zipcode.trim().isEmpty()) {
                zipcode = null;
            }
```

```
        }

        this.dsl.update(CUSTOMER)
                .set(CUSTOMER.NAME, name)
                .set(CUSTOMER.ADDRESS, address)
                .set(CUSTOMER.ZIPCODE, zipcode)
                .set(CUSTOMER.PHONE, phone)
                .where(CUSTOMER.CUSTOMER_ID.eq(customerId))
                .execute();
    }

    /**
     * Удаление заказчика
     *
     * @param customerId
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
    public void delete(int customerId) {
        this.dsl.deleteFrom(CUSTOMER)
                .where(CUSTOMER.CUSTOMER_ID.eq(customerId))
                .execute();
    }
}
```

Теперь перейдём к написанию контроллера. Классы контроллеров начинаются с аннотации @Controller. Для определения действий контроллера необходимо добавить аннотацию @RequestMapping перед методом и указать в ней маршрут, по которому будет вызвано действие контроллера. Маршрут указывается в параметре value. В параметре method можно указать метод HTTP запроса (PUT, GET, POST, DELETE). Входной точкой нашего контроллера будет метод index, он отвечает за отображение JSP страницы (представления). Эта страница содержит разметку для отображения грида, панель инструментов и навигации.

Данные для отображения загружаются асинхронно компонентом jqGrid (маршрут /customer/getdata). С данным маршрутом связан метод getData. Метод содержит дополнительную аннотацию @ResponseBody, которая говорит о том, что наш метод возвращает объект для сериализации в один из форматов. В аннотации @RequestMapping задан параметр produces = MediaType.APPLICATION_JSON, что обозначает, что возвращаемый объект будет сериализован в формат JSON. Именно в этом методе мы работаем с классом `JqGridCustomer` описанном выше. Аннотация @RequestParam позволяет извлечь значение параметра из HTTP запроса. Данный метод класса работает с GET запросами. Параметр value в аннотации @RequestParam задаёт имя параметра HTTP запроса для извлечения. Параметр required задаёт, является ли параметр HTTP запроса обязательным. Параметр defaultValue задаёт значение по умолчанию, которое будет подставлено в случае отсутствия HHTP параметра.

Метод addCustomer предназначен для добавления нового заказчика. Он связан с маршрутом /customer/create, и в отличие от предыдущего метода работает с POST запросом. Метод возвращает {success: true} в случае успешного добавления, и объект с текстом ошибки в случае ошибки. Данный метод работает с классом бизнес слоя `CustomerManager`.

Метод editCustomer связан с маршрутом /customer/edit и предназначен для редактирования заказчика. Метод deleteCustomer связан с маршрутом /customer/delete и предназначен для удаления заказчика.

```
package ru.ibase.fbjavaex.controllers;

import java.util.HashMap;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestParam;
import javax.ws.rs.core.MediaType;

import org.springframework.beans.factory.annotation.Autowired;

import ru.ibase.fbjavaex.managers.CustomerManager;

import ru.ibase.fbjavaex.jqgrid.JqGridCustomer;
import ru.ibase.fbjavaex.jqgrid.JqGridData;


/**
 * Контроллер заказчиков
 *
 * @author Simonov Denis
 */
@Controller
public class CustomerController {


    @Autowired(required = true)
    private JqGridCustomer customerGrid;

    @Autowired(required = true)
    private CustomerManager customerManager;


    /**
     * Действие по умолчанию
     * Возвращает имя JSP страницы (представления) для отображения
     *
     * @param map
     * @return имя JSP шаблона
     */
    @RequestMapping(value = "/customer/", method = RequestMethod.GET)
    public String index(ModelMap map) {
        return "customer";
    }

    /**
     * Возвращает данные в формате JSON для jqGrid
     *
     * @param rows количество строк на страницу
     * @param page номер страницы
     * @param sIdx поле для сортировки
     * @param sOrd порядок сортировки
     * @param search должен ли осуществляться поиск
     * @param searchField поле поиска
     * @param searchString значение поиска
     * @param searchOper операция поиска
     * @return JSON для jqGrid
```

```
     */
    @RequestMapping(value = "/customer/getdata",
            method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public JqGridData getData(
            // количество записей на странице
            @RequestParam(value = "rows", required = false, defaultValue = "20") int rows,
            // номер текущей страницы
            @RequestParam(value = "page", required = false, defaultValue = "1") int page,
            // поле для сортировки
            @RequestParam(value = "sidx", required = false, defaultValue = "") String sIdx,
            // направление сортировки
            @RequestParam(value = "sord", required = false, defaultValue = "asc") String sOrd,
            // осуществляется ли поиск
            @RequestParam(value = "_search", required = false, defaultValue = "false") Boolean
search,
            // поле поиска
            @RequestParam(value = "searchField", required = false, defaultValue = "") String
searchField,
            // значение поиска
            @RequestParam(value = "searchString", required = false, defaultValue = "") String
searchString,
            // операция поиска
            @RequestParam(value = "searchOper", required = false, defaultValue = "") String
searchOper,
            // фильтр
            @RequestParam(value="filters", required=false, defaultValue="") String filters) {
        customerGrid.setLimit(rows);
        customerGrid.setPageNo(page);
        customerGrid.setOrderBy(sIdx, sOrd);
        if (search) {
            customerGrid.setSearchCondition(searchField, searchString, searchOper);
        }

        return customerGrid.getJqGridData();
    }

    @RequestMapping(value = "/customer/create",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> addCustomer(
            @RequestParam(value = "NAME", required = true, defaultValue = "") String name,
            @RequestParam(value = "ADDRESS", required = false, defaultValue = "") String address,
            @RequestParam(value = "ZIPCODE", required = false, defaultValue = "") String zipcode,
            @RequestParam(value = "PHONE", required = false, defaultValue = "") String phone) {
        Map<String, Object> map = new HashMap<>();
        try {
            customerManager.create(name, address, zipcode, phone);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    @RequestMapping(value = "/customer/edit",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> editCustomer(
            @RequestParam(value = "CUSTOMER_ID", required = true, defaultValue = "0") int
customerId,
            @RequestParam(value = "NAME", required = true, defaultValue = "") String name,
            @RequestParam(value = "ADDRESS", required = false, defaultValue = "") String address,
            @RequestParam(value = "ZIPCODE", required = false, defaultValue = "") String zipcode,
            @RequestParam(value = "PHONE", required = false, defaultValue = "") String phone) {
        Map<String, Object> map = new HashMap<>();
        try {
```

```
            customerManager.edit(customerId, name, address, zipcode, phone);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    @RequestMapping(value = "/customer/delete",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> deleteCustomer(
            @RequestParam(value = "CUSTOMER_ID", required = true, defaultValue = "0") int
customerId) {
        Map<String, Object> map = new HashMap<>();
        try {
            customerManager.delete(customerId);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }
}
```

JSP страница для отображения справочника заказчиков не содержит ничего особенного: разметку с основными частями страницы, таблицу для отображения грида и блок для отображения панели навигации. JSP шаблоны не очень продвинутое средство, при желании вы можете заменить их на другие системы шаблонов, которые поддерживают наследование. В файле ../jspf/head.jspf содержатся общие скрипты и стили для всех страниц сайта, а файл ../jspf/menu.jspf главное меню сайта. Мы не будем приводить их код, он довольно простой и при желании вы можете посмотреть его в исходных кодах проекта.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="cp" value="${pageContext.request.servletContext.contextPath}" scope="request" />

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Пример Spring MVC приложения с использованием Firebird и jOOQ</title>

        <!-- Скрипты и стили -->
        <%@ include file="../jspf/head.jspf" %>
        <script src="${cp}/resources/js/jqGridCustomer.js"></script>
    </head>
    <body>
        <!-- Навигационное меню -->
        <%@ include file="../jspf/menu.jspf" %>

        <div class="container body-content">

            <h2>Customers</h2>

            <table id="jqGridCustomer"></table>
            <div id="jqPagerCustomer"></div>

            <hr/>
            <footer>
                <p>&copy; 2016 - Пример Spring MVC приложения с использованием Firebird и
jOOQ</p>
            </footer>
        </div>
```

```
<script type="text/javascript">
    $(document).ready(function () {
        JqGridCustomer({
            baseAddress: '${cp}'
        });
    });
</script>

    </body>
</html>
```

Основная логика на стороне клиента сосредоточена в JavaScript модуле */resources/js/jqGridCustomer.js*

```
var JqGridCustomer = (function ($) {

    return function (options) {
        var jqGridCustomer = {
            dbGrid: null,
            // опции
            options: $.extend({
                baseAddress: null,
                showEditorPanel: true
            }, options),
            // возвращает модель
            getColModel: function () {
                return [
                    {
                        label: 'Id', // подпись
                        name: 'CUSTOMER_ID', // имя поля
                        key: true, // признак ключевого поля
                        hidden: true          // скрыт
                    },
                    {
                        label: 'Name', // подпись поля
                        name: 'NAME', // имя поля
                        width: 240, // ширина
                        sortable: true, // разрешена сортировка
                        editable: true, // разрешено редактирование
                        edittype: "text", // тип поля в редакторе
                        search: true, // разрешён поиск
                        searchoptions: {
                            sopt: ['eq', 'bw', 'cn'] // разрешённые операторы поиска
                        },
                        editoptions: {size: 30, maxlength: 60}, // размер и максимальная длина
для поля ввода
                        editrules: {required: true}      // говорит о том, что поле обязательное
                    },
                    {
                        label: 'Address',
                        name: 'ADDRESS',
                        width: 300,
                        sortable: false, // запрещаем сортировку
                        editable: true, // редактируемое
                        search: false, // запрещаем поиск
                        edittype: "textarea", // мемо поле
                        editoptions: {maxlength: 250, cols: 30, rows: 4}
                    },
                    {
                        label: 'Zip Code',
                        name: 'ZIPCODE',
                        width: 30,
                        sortable: false,
                        editable: true,
                        search: false,
                        edittype: "text",
                        editoptions: {size: 30, maxlength: 10}
                    },
```

```
                {
                    label: 'Phone',
                    name: 'PHONE',
                    width: 80,
                    sortable: false,
                    editable: true,
                    search: false,
                    edittype: "text",
                    editoptions: {size: 30, maxlength: 14}
                }
            ];
        },
        // инициализация грида
        initGrid: function () {
            // url для получения данных
            var url = jqGridCustomer.options.baseAddress + '/customer/getdata';
            jqGridCustomer.dbGrid = $("#jqGridCustomer").jqGrid({
                url: url,
                datatype: "json", // формат получения данных
                mtype: "GET", // тип http запроса
                colModel: jqGridCustomer.getColModel(),
                rowNum: 500, // число отображаемых строк
                loadonce: false, // загрузка только один раз
                sortname: 'NAME', // сортировка по умолчанию по столбцу NAME
                sortorder: "asc", // порядок сортировки
                width: window.innerWidth - 80, // ширина грида
                height: 500, // высота грида
                viewrecords: true, // отображать количество записей
                guiStyle: "bootstrap",
                iconSet: "fontAwesome",
                caption: "Customers", // подпись к гриду
                // элемент для отображения навигации
                pager: 'jqPagerCustomer'
            });
        },
        // опции редактирования
        getEditOptions: function () {
            return {
                url: jqGridCustomer.options.baseAddress + '/customer/edit',
                reloadAfterSubmit: true,
                closeOnEscape: true,
                closeAfterEdit: true,
                drag: true,
                width: 400,
                afterSubmit: jqGridCustomer.afterSubmit,
                editData: {
                    // дополнительно к значениям из формы передаём ключевое поле
                    CUSTOMER_ID: function () {
                        // получаем текущую строку
                        var selectedRow = jqGridCustomer.dbGrid.getGridParam("selrow");
                        // получаем значение интересующего нас поля
                        var value = jqGridCustomer.dbGrid.getCell(selectedRow,
'CUSTOMER_ID');
                        return value;
                    }
                }
            };
        },
        // опции добавления
        getAddOptions: function () {
            return {
                url: jqGridCustomer.options.baseAddress + '/customer/create',
                reloadAfterSubmit: true,
                closeOnEscape: true,
                closeAfterAdd: true,
                drag: true,
                width: 400,
                afterSubmit: jqGridCustomer.afterSubmit
            };
        },
        // опции удаления
        getDeleteOptions: function () {
```

```
                return {
                    url: jqGridCustomer.options.baseAddress + '/customer/delete',
                    reloadAfterSubmit: true,
                    closeOnEscape: true,
                    closeAfterDelete: true,
                    drag: true,
                    msg: "Удалить выделенного заказчика?",
                    afterSubmit: jqGridCustomer.afterSubmit,
                    delData: {
                        // передаём ключевое поле
                        CUSTOMER_ID: function () {
                            var selectedRow = jqGridCustomer.dbGrid.getGridParam("selrow");
                            var value = jqGridCustomer.dbGrid.getCell(selectedRow,
'CUSTOMER_ID');
                            return value;
                        }
                    }
                };
            },
            // инициализация панели навигации вместе с диалогами редактирования
            initPagerWithEditors: function () {
                jqGridCustomer.dbGrid.jqGrid('navGrid', '#jqPagerCustomer',
                        {
                            // кнопки
                            search: true, // поиск
                            add: true, // добавление
                            edit: true, // редактирование
                            del: true, // удаление
                            view: true, // просмотр записи
                            refresh: true, // обновление
                            // подписи кнопок
                            searchtext: "Поиск",
                            addtext: "Добавить",
                            edittext: "Изменить",
                            deltext: "Удалить",
                            viewtext: "Смотреть",
                            viewtitle: "Выбранная запись",
                            refreshtext: "Обновить"
                        },
                        jqGridCustomer.getEditOptions(),
                        jqGridCustomer.getAddOptions(),
                        jqGridCustomer.getDeleteOptions()
                        );
            },
            // инициализация панели навигации вместе без диалогов редактирования
            initPagerWithoutEditors: function () {
                jqGridCustomer.dbGrid.jqGrid('navGrid', '#jqPagerCustomer',
                        {
                            // кнопки
                            search: true, // поиск
                            add: false, // добавление
                            edit: false, // редактирование
                            del: false, // удаление
                            view: false, // просмотр записи
                            refresh: true, // обновление
                            // подписи кнопок
                            searchtext: "Поиск",
                            viewtext: "Смотреть",
                            viewtitle: "Выбранная запись",
                            refreshtext: "Обновить"
                        }
                );
            },
            // инициализация панели навигации
            initPager: function () {
                if (jqGridCustomer.options.showEditorPanel) {
                    jqGridCustomer.initPagerWithEditors();
                } else {
                    jqGridCustomer.initPagerWithoutEditors();
                }
            },
```

```
            // инициализация
            init: function () {
                jqGridCustomer.initGrid();
                jqGridCustomer.initPager();
            },
            // обработчик результатов обработки форм (операций)
            afterSubmit: function (response, postdata) {
                var responseData = response.responseJSON;
                // проверяем результат на наличие сообщений об ошибках
                if (responseData.hasOwnProperty("error")) {
                    if (responseData.error.length) {
                        return [false, responseData.error];
                    }
                } else {
                    // если не была возвращена ошибка обновляем грид
                    $(this).jqGrid(
                            'setGridParam',
                            {
                                datatype: 'json'
                            }
                    ).trigger('reloadGrid');
                }
                return [true, "", 0];
            }
        };
        jqGridCustomer.init();
        return jqGridCustomer;
    };
})(jQuery);
```

Сетка jqGrid создаётся в методе initGrid и привязывается к html элементу с идентификатором jqGridCustomer. Описание столбцов (колонок) грида возвращается методом getColModel. Каждый столбец в jqGrid имеет достаточно много возможных свойств. В исходном коде присутствуют комментарии, объясняющие свойства столбцов. Подробнее о конфигурировании модели столбцов jqGrid вы можете прочитать в документации проекта jqGrid в разделе ColModel API.

Панель навигации может быть создана с кнопками редактирования или без них (методы `initPagerWithEditors` и `initPagerWithoutEditors` соответственно). Конструктор панели прикрепляет её к элементу с идентификатором jqPagerCustomer. Опции создания панели навигации описаны в разделе Navigator документации jqGrid.

Функции `getEditOptions`, `getAddOptions`, `getDeleteOptions` возвращают опции диалогов редактирования, добавления и удаления соответственно. Свойство url указывает, по какому адресу будут отправлены данные после нажатия кнопки OK в диалоге. Свойство `afterSubmit` – событие, происходящее после отправки данных на сервер и получения ответа от него. В методе `afterSubmit` проверяется, не вернул ли наш контроллер ошибку. Если ошибки не было, то производится обновление грида, в противном случае ошибка сообщается пользователю. Обратите внимание на свойство `editData`. Оно позволяет задать значения дополнительных полей, которые не участвуют в диалоге редактирования. Дело в том, что диалоги редактирования не включают в себя значение скрытых полей, а отображать автоматически генерируемые ключи не сильно хочется.

Создание журналов

В отличие от справочников журналы содержат довольно большое количество записей и являются часто пополняемыми. Большинство журналов содержат поле с датой создания документа. Чтобы уменьшить количество выбираемых данных обычно принято вводить такое понятие как рабочий период для того, чтобы уменьшить объём данных передаваемый на клиента. Рабочий период – это диапазон дат, внутри которого требуются рабочие документы. Рабочий период описывается классом WorkingPeriod. Этот класс создаётся через бин workingPeriod в классе конфигурации ru.ibase.fbjavaex.config.JooqConfig.

```
package ru.ibase.fbjavaex.config;

import java.sql.Timestamp;
import java.time.LocalDateTime;

/**
 * Рабочий период
 *
 * @author Simonov Denis
 */
public class WorkingPeriod {

    private Timestamp beginDate;
    private Timestamp endDate;

    /**
     * Конструктор
     */
    WorkingPeriod() {
        // в реальных приложениях вычисляется от текущей даты
        this.beginDate = Timestamp.valueOf("2015-06-01 00:00:00");
        this.endDate = Timestamp.valueOf(LocalDateTime.now().plusDays(1));
    }

    /**
     * Возвращает дату начала рабочего периода
     *
     * @return
     */
    public Timestamp getBeginDate() {
        return this.beginDate;
    }

    /**
     * Возвращает дату окончания рабочего периода
     *
     * @return
     */
    public Timestamp getEndDate() {
        return this.endDate;
    }

    /**
     * Установка даты начала рабочего периода
     *
     * @param value
     */
    public void setBeginDate(Timestamp value) {
        this.beginDate = value;
    }

    /**
```

```
     * Установка даты окончания рабочего периода
     *
     * @param value
     */
    public void setEndDate(Timestamp value) {
        this.endDate = value;
    }

    /**
     * Установка рабочего периода
     *
     * @param beginDate
     * @param endDate
     */
    public void setRangeDate(Timestamp beginDate, Timestamp endDate) {
        this.beginDate = beginDate;
        this.endDate = endDate;
    }
}
```

В нашем приложении будет один журнал «Счёт-фактуры». Счёт-фактура – состоит из заголовка, где описываются общие атрибуты (номер, дата, заказчик …), и позиций счёт-фактуры (наименование товара, количество, стоимостью и т.д.). Шапка счёт-фактуру отображается в основной сетке, а позиции могут быть просмотрены в детализирующей сетке, которая раскрывается по щелчку по значку «+» на нужном документе.

Реализуем класс для просмотра шапок счёт-фактуры через jqGrid, он будет наследоваться от нашего абстрактного класса `ru.ibase.fbjavaex.jqgrid.JqGrid`, описанного выше. В нём имеется возможность поиска наименованию заказчика и дате счёта. Кроме того данный класс поддерживает сортировку по дате в обоих направлениях.

```
package ru.ibase.fbjavaex.jqgrid;

import java.sql.*;
import org.jooq.*;

import java.util.List;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import ru.ibase.fbjavaex.config.WorkingPeriod;

import static ru.ibase.fbjavaex.exampledb.Tables.INVOICE;
import static ru.ibase.fbjavaex.exampledb.Tables.CUSTOMER;

/**
 * Обработчик грида для журнала счёт-фактур
 *
 * @author Simonov Denis
 */
public class JqGridInvoice extends JqGrid {

    @Autowired(required = true)
    private WorkingPeriod workingPeriod;

    /**
     * Добавление условия поиска
     *
     * @param query
     */
```

```
    private void makeSearchCondition(SelectQuery<?> query) {
        // добавлением в запрос условия поиска, если он производится
        // для разных полей доступны разные операторы
        // сравнения при поиске
        if (this.searchString.isEmpty()) {
            return;
        }

        if (this.searchField.equals("CUSTOMER_NAME")) {
            switch (this.searchOper) {
                case "eq": // equal
                    query.addConditions(CUSTOMER.NAME.eq(this.searchString));
                    break;
                case "bw": // starting with
                    query.addConditions(CUSTOMER.NAME.startsWith(this.searchString));
                    break;
                case "cn": // containing
                    query.addConditions(CUSTOMER.NAME.contains(this.searchString));
                    break;
            }
        }
        if (this.searchField.equals("INVOICE_DATE")) {
            Timestamp dateValue = Timestamp.valueOf(this.searchString);

            switch (this.searchOper) {
                case "eq": // =
                    query.addConditions(INVOICE.INVOICE_DATE.eq(dateValue));
                    break;
                case "lt": // <
                    query.addConditions(INVOICE.INVOICE_DATE.lt(dateValue));
                    break;
                case "le": // <=
                    query.addConditions(INVOICE.INVOICE_DATE.le(dateValue));
                    break;
                case "gt": // >
                    query.addConditions(INVOICE.INVOICE_DATE.gt(dateValue));
                    break;
                case "ge": // >=
                    query.addConditions(INVOICE.INVOICE_DATE.ge(dateValue));
                    break;

            }
        }
    }

    /**
     * Возвращает общее количество записей
     *
     * @return
     */
    @Override
    public int getCountRecord() {
        SelectFinalStep<?> select
                = dsl.selectCount()
                        .from(INVOICE)
                        .where(INVOICE.INVOICE_DATE.between(this.workingPeriod.getBeginDate(),
this.workingPeriod.getEndDate()));

        SelectQuery<?> query = select.getQuery();

        if (this.searchFlag) {
            makeSearchCondition(query);
        }

        return (int) query.fetch().getValue(0, 0);
    }



    /**
```

```
     * Возвращает список счёт-фактур
     *
     * @return
     */
    @Override
    public List<Map<String, Object>> getRecords() {
        SelectFinalStep<?> select
                = dsl.select(
                        INVOICE.INVOICE_ID,
                        INVOICE.CUSTOMER_ID,
                        CUSTOMER.NAME.as("CUSTOMER_NAME"),
                        INVOICE.INVOICE_DATE,
                        INVOICE.PAID,
                        INVOICE.TOTAL_SALE)
                        .from(INVOICE)
                        .innerJoin(CUSTOMER).on(CUSTOMER.CUSTOMER_ID.eq(INVOICE.CUSTOMER_ID))
                        .where(INVOICE.INVOICE_DATE.between(this.workingPeriod.getBeginDate(),
this.workingPeriod.getEndDate()));

        SelectQuery<?> query = select.getQuery();
        // добавляем условие поиска
        if (this.searchFlag) {
            makeSearchCondition(query);
        }
        // Добавляем сортировку
        if (this.sIdx.equals("INVOICE_DATE")) {
            switch (this.sOrd) {
                case "asc":
                    query.addOrderBy(INVOICE.INVOICE_DATE.asc());
                    break;
                case "desc":
                    query.addOrderBy(INVOICE.INVOICE_DATE.desc());
                    break;
            }
        }
        // Добавляем ограничение и смещение
        if (this.limit != 0) {
            query.addLimit(this.limit);
        }
        if (this.offset != 0) {
            query.addOffset(this.offset);
        }

        return query.fetchMaps();
    }
}
```

Класс для просмотра позиций счёт-фактуры через jqGrid несколько проще. Во-первых, его записи отфильтрованы по коду шапки счёт фактуры, а во-вторых в нём мы не будем реализовывать поиск и пользовательскую сортировку.

```
package ru.ibase.fbjavaex.jqgrid;

import org.jooq.*;

import java.util.List;
import java.util.Map;

import static ru.ibase.fbjavaex.exampledb.Tables.INVOICE_LINE;
import static ru.ibase.fbjavaex.exampledb.Tables.PRODUCT;

/**
 * Обработчик грида для позиций журнала счёт-фактур
 *
 * @author Simonov Denis
 */
public class JqGridInvoiceLine extends JqGrid {
```

```
    private int invoiceId;


    public int getInvoiceId() {
        return this.invoiceId;
    }

    public void setInvoiceId(int invoiceId) {
        this.invoiceId = invoiceId;
    }

    /**
     * Возвращает общее количество записей
     *
     * @return
     */
    @Override
    public int getCountRecord() {
        SelectFinalStep<?> select
                = dsl.selectCount()
                        .from(INVOICE_LINE)
                        .where(INVOICE_LINE.INVOICE_ID.eq(this.invoiceId));

        SelectQuery<?> query = select.getQuery();


        return (int) query.fetch().getValue(0, 0);
    }


    /**
     * Возвращает позиции накладной
     *
     * @return
     */
    @Override
    public List<Map<String, Object>> getRecords() {
        SelectFinalStep<?> select
                = dsl.select(
                        INVOICE_LINE.INVOICE_LINE_ID,
                        INVOICE_LINE.INVOICE_ID,
                        INVOICE_LINE.PRODUCT_ID,
                        PRODUCT.NAME.as("PRODUCT_NAME"),
                        INVOICE_LINE.QUANTITY,
                        INVOICE_LINE.SALE_PRICE,
                        INVOICE_LINE.SALE_PRICE.mul(INVOICE_LINE.QUANTITY).as("TOTAL"))
                        .from(INVOICE_LINE)
                        .innerJoin(PRODUCT).on(PRODUCT.PRODUCT_ID.eq(INVOICE_LINE.PRODUCT_ID))
                        .where(INVOICE_LINE.INVOICE_ID.eq(this.invoiceId));

        SelectQuery<?> query = select.getQuery();
        return query.fetchMaps();
    }
}
```

Добавлять, редактировать, удалять счёт фактуры (и их позиции), а также оплачивать их, мы будем через класс ru.ibase.fbjavaex.managers.InvoiceManager, который является своеобразным бизнес слоем. Все операции в этом слое мы будем осуществлять в транзакции с уровнем изолированности Snapshot. В этом классе все операции с базой данных осуществляются с помощью вызовов хранимых процедур (это не является обязательным, просто показан один из вариантов).

```java
package ru.ibase.fbjavaex.managers;


import java.sql.Timestamp;
import org.jooq.DSLContext;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Isolation;

import static ru.ibase.fbjavaex.exampledb.Sequences.GEN_INVOICE_ID;
import static ru.ibase.fbjavaex.exampledb.Routines.spAddInvoice;
import static ru.ibase.fbjavaex.exampledb.Routines.spEditInvoice;
import static ru.ibase.fbjavaex.exampledb.Routines.spPayForInovice;
import static ru.ibase.fbjavaex.exampledb.Routines.spDeleteInvoice;
import static ru.ibase.fbjavaex.exampledb.Routines.spAddInvoiceLine;
import static ru.ibase.fbjavaex.exampledb.Routines.spEditInvoiceLine;
import static ru.ibase.fbjavaex.exampledb.Routines.spDeleteInvoiceLine;

/**
 * Менеджер счёт фактур
 *
 * @author Simonov Denis
 */
public class InvoiceManager {

    @Autowired(required = true)
    private DSLContext dsl;

    /**
     * Добавление шапки счёт фактуры
     *
     * @param customerId
     * @param invoiceDate
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
    public void create(Integer customerId, Timestamp invoiceDate) {
        int invoiceId = this.dsl.nextval(GEN_INVOICE_ID).intValue();

        spAddInvoice(this.dsl.configuration(),
                invoiceId,
                customerId,
                invoiceDate);
    }

    /**
     * Редактирование счёт фактуры
     *
     * @param invoiceId
     * @param customerId
     * @param invoiceDate
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
    public void edit(Integer invoiceId, Integer customerId, Timestamp invoiceDate) {
        spEditInvoice(this.dsl.configuration(),
                invoiceId,
                customerId,
                invoiceDate);
    }

    /**
     * Оплата счёт фактуры
     *
     * @param invoiceId
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
```

```
    public void pay(Integer invoiceId) {
        spPayForInovice(this.dsl.configuration(),
                invoiceId);
    }

    /**
     * Удаление счёт фактуры
     *
     * @param invoiceId
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
    public void delete(Integer invoiceId) {
        spDeleteInvoice(this.dsl.configuration(),
                invoiceId);
    }

    /**
     * Добавление позиции счёт фактуры
     *
     * @param invoiceId
     * @param productId
     * @param quantity
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
    public void addInvoiceLine(Integer invoiceId, Integer productId, Integer quantity) {
        spAddInvoiceLine(this.dsl.configuration(),
                invoiceId,
                productId,
                quantity);
    }

    /**
     * Редактирование позиции счёт фактуры
     *
     * @param invoiceLineId
     * @param quantity
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
    public void editInvoiceLine(Integer invoiceLineId, Integer quantity) {
        spEditInvoiceLine(this.dsl.configuration(),
                invoiceLineId,
                quantity);
    }

    /**
     * Удаление позиции счёт фактуры
     *
     * @param invoiceLineId
     */
    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.REPEATABLE_READ)
    public void deleteInvoiceLine(Integer invoiceLineId) {
        spDeleteInvoiceLine(this.dsl.configuration(),
                invoiceLineId);
    }

}
```

Теперь перейдём к написанию контроллера. Входной точкой нашего контроллера будет метод index, он отвечает за отображение JSP страницы (представления). Эта страница содержит разметку для отображения грида, панель инструментов и навигации.

Данные для отображения шапок счёт фактуры загружаются асинхронно компонентом jqGrid (маршрут /invoice/getdata). С данным маршрутом связан метод getData (аналогично справочникам). Позиции счёт фактуры возвращаются методом getDetailData (маршрут /invoice/getdetaildata). В этот метод передаётся код счёт фактуры, на которой был раскрыт детализирующий грид. Методы addInvoice, editInvoice, payInvoice, deleteInvoice осуществляют добавление, редактирование, оплату и удаление счёт фактуры. Методы addInvoiceLine, editInvoiceLine, deleteInvoiceLine осуществляют добавление, редактирование и удаление позиции счёт фактуры.

```
package ru.ibase.fbjavaex.controllers;

import java.sql.Timestamp;
import java.util.HashMap;
import java.util.Map;
import java.util.Date;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.beans.PropertyEditorSupport;

import javax.ws.rs.core.MediaType;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.WebDataBinder;
import ru.ibase.fbjavaex.jqgrid.JqGridInvoice;
import ru.ibase.fbjavaex.jqgrid.JqGridInvoiceLine;

import ru.ibase.fbjavaex.managers.InvoiceManager;

import ru.ibase.fbjavaex.jqgrid.JqGridData;


/**
 * Контроллер счёт-фактур
 *
 * @author Simonov Denis
 */
@Controller
public class InvoiceController {

    @Autowired(required = true)
    private JqGridInvoice invoiceGrid;

    @Autowired(required = true)
    private JqGridInvoiceLine invoiceLineGrid;

    @Autowired(required = true)
    private InvoiceManager invoiceManager;

    /**
     * Описываем, как преобразуется строка в дату
     * из входных параметров HTTP запроса
     *
     * @param binder
     */
    @InitBinder
    public void initBinder(WebDataBinder binder)   {
        binder.registerCustomEditor(Timestamp.class,
                new PropertyEditorSupport() {
            @Override
```

```
            public void setAsText(String value) {
                try {
                    if ((value == null) || (value.isEmpty())) {
                        setValue(null);
                    } else {
                        Date parsedDate = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss").parse(value);
                        setValue(new Timestamp(parsedDate.getTime()));
                    }
                } catch (ParseException e) {
                    throw new java.lang.IllegalArgumentException(value);
                }
            }
        });
    }

    /**
     * Действие по умолчанию
     * Возвращает имя JSP страницы (представления) для отображения
     *
     * @param map
     * @return имя JSP страницы
     */
    @RequestMapping(value = "/invoice/", method = RequestMethod.GET)
    public String index(ModelMap map) {

        return "invoice";
    }

    /**
     * Возвращает список счёт фактур в формате JSON для jqGrid
     *
     * @param rows количество записей на странице
     * @param page номер текущей страницы
     * @param sIdx поле сортировки
     * @param sOrd порядок сортировки
     * @param search флаг поиска
     * @param searchField поле поиска
     * @param searchString значение поиска
     * @param searchOper операция сравнения
     * @param filters фильтр
     * @return
     */
    @RequestMapping(value = "/invoice/getdata",
            method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public JqGridData getData(
            @RequestParam(value = "rows", required = false, defaultValue = "20") int rows,
            @RequestParam(value = "page", required = false, defaultValue = "1") int page,
            @RequestParam(value = "sidx", required = false, defaultValue = "") String sIdx,
            @RequestParam(value = "sord", required = false, defaultValue = "asc") String sOrd,
            @RequestParam(value = "_search", required = false, defaultValue = "false") Boolean
search,
            @RequestParam(value = "searchField", required = false, defaultValue = "") String
searchField,
            @RequestParam(value = "searchString", required = false, defaultValue = "") String
searchString,
            @RequestParam(value = "searchOper", required = false, defaultValue = "") String
searchOper,
            @RequestParam(value = "filters", required = false, defaultValue = "") String filters)
{

        if (search) {
            invoiceGrid.setSearchCondition(searchField, searchString, searchOper);
        }
        invoiceGrid.setLimit(rows);
        invoiceGrid.setPageNo(page);

        invoiceGrid.setOrderBy(sIdx, sOrd);

        return invoiceGrid.getJqGridData();
```

```
    }

    /**
     * Добавляет счёт фактуру
     *
     * @param customerId код заказчика
     * @param invoiceDate дата счёт фактуры
     * @return
     */
    @RequestMapping(value = "/invoice/create",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> addInvoice(
            @RequestParam(value = "CUSTOMER_ID", required = true, defaultValue = "0") Integer
customerId,
            @RequestParam(value = "INVOICE_DATE", required = false, defaultValue = "") Timestamp
invoiceDate) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.create(customerId, invoiceDate);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Редактирует счёт фактуру
     *
     * @param invoiceId код счёт фактуры
     * @param customerId код заказчика
     * @param invoiceDate дата счёт фактуры
     * @return
     */
    @RequestMapping(value = "/invoice/edit",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> editInvoice(
            @RequestParam(value = "INVOICE_ID", required = true, defaultValue = "0") Integer
invoiceId,
            @RequestParam(value = "CUSTOMER_ID", required = true, defaultValue = "0") Integer
customerId,
            @RequestParam(value = "INVOICE_DATE", required = false, defaultValue = "") Timestamp
invoiceDate) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.edit(invoiceId, customerId, invoiceDate);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Оплачивает счёт фактуру
     *
     * @param invoiceId код счёт фактуры
     * @return
     */
    @RequestMapping(value = "/invoice/pay",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> payInvoice(
            @RequestParam(value = "INVOICE_ID", required = true, defaultValue = "0") Integer
invoiceId) {
        Map<String, Object> map = new HashMap<>();
        try {
```

```
            invoiceManager.pay(invoiceId);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Удаляет счёт фактуру
     *
     * @param invoiceId код счёт фактуры
     * @return
     */
    @RequestMapping(value = "/invoice/delete",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> deleteInvoice(
            @RequestParam(value = "INVOICE_ID", required = true, defaultValue = "0") Integer
invoiceId) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.delete(invoiceId);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Возвращает список позиций счёт фактуры
     *
     * @param invoice_id код счёт фактуры
     * @return
     */
    @RequestMapping(value = "/invoice/getdetaildata",
            method = RequestMethod.GET,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public JqGridData getDetailData(
            @RequestParam(value = "INVOICE_ID", required = true) int invoice_id) {

        invoiceLineGrid.setInvoiceId(invoice_id);

        return invoiceLineGrid.getJqGridData();

    }

    /**
     * Добавляет позицию счёт фактуры
     *
     * @param invoiceId код счёт фактуры
     * @param productId код товара
     * @param quantity количество товара
     * @return
     */
    @RequestMapping(value = "/invoice/createdetail",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> addInvoiceLine(
            @RequestParam(value = "INVOICE_ID", required = true, defaultValue = "0") Integer
invoiceId,
            @RequestParam(value = "PRODUCT_ID", required = true, defaultValue = "0") Integer
productId,
            @RequestParam(value = "QUANTITY", required = true, defaultValue = "0") Integer
quantity) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.addInvoiceLine(invoiceId, productId, quantity);
```

```
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Редактирует позицию счёт фактуры
     *
     * @param invoiceLineId код позиции счёт фактуры
     * @param quantity количество товара
     * @return
     */
    @RequestMapping(value = "/invoice/editdetail",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> editInvoiceLine(
            @RequestParam(value = "INVOICE_LINE_ID", required = true, defaultValue = "0") Integer
invoiceLineId,
            @RequestParam(value = "QUANTITY", required = true, defaultValue = "0") Integer
quantity) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.editInvoiceLine(invoiceLineId, quantity);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }

    /**
     * Удаляет позицию счёт фактуры
     *
     * @param invoiceLineId код позиции счёт фактуры
     * @return
     */
    @RequestMapping(value = "/invoice/deletedetail",
            method = RequestMethod.POST,
            produces = MediaType.APPLICATION_JSON)
    @ResponseBody
    public Map<String, Object> deleteInvoiceLine(
            @RequestParam(value = "INVOICE_LINE_ID", required = true, defaultValue = "0") Integer
invoiceLineId) {
        Map<String, Object> map = new HashMap<>();
        try {
            invoiceManager.deleteInvoiceLine(invoiceLineId);
            map.put("success", true);
        } catch (Exception ex) {
            map.put("error", ex.getMessage());
        }
        return map;
    }
}
```

В целом, контроллер счёт фактур похож на контроллеры справочников за двумя исключениями:

1. Контроллер отображает и работает с данными, как главного, так и детализирующего грида.
2. Счёт фактуры отфильтрованы по полю дата так, чтобы в выборку попадали только те счёт фактуры, которые входят в рабочий период.

При работе с датами существует много особенностей.

Тип java.sql.Timestamp в Java поддерживает точность до наносекунд, в то время как в Firebird максимальная точность типа TIMESTAMP составляет десятитысячную долю секунды. На самом деле это не является большой проблемой.

Типы даты и времени в Java поддерживают работу временными зонами. С другой стороны, в настоящее время Firebird не поддерживает тип TIMESTAMP WITH TIMEZONE. В этом случае Java считает, что даты в базе данных хранятся в часовом поясе сервера (а не в UTC как вы могли бы подумать). Однако при сериализации в JSON время будет преобразовано в UTC. Это надо учитывать при обработке времени в JavaScript.

Внимание!

Java берёт смещение времени из собственной базы временных зон, а не из операционной системы. Это обстоятельство существенно повышает требования к актуальности версии JDK. Если у вас установлена древняя JDK, то работа с датой и временем может вестись не верно.

По умолчанию дата сериализуется в JSON в числовом представлении (как число наносекунд прошедших с 1 января 1970). Это не всегда удобно. Для того чтобы дата сериализовалась в текстовом представлении в классе WebAppConfig, описанном выше, в методе configureMessageConverters свойству конфигурации SerializationFeature.WRITE_DATES_AS_TIMESTAMPS преобразования даты указать значение false.

```
@Configuration
@ComponentScan("ru.ibase.fbjavaex")
@EnableWebMvc
public class WebAppConfig extends WebMvcConfigurerAdapter {

    @Override
    public void configureMessageConverters(List<HttpMessageConverter<?>> httpMessageConverters) {
        MappingJackson2HttpMessageConverter jsonConverter = new
MappingJackson2HttpMessageConverter();
        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false);
        jsonConverter.setObjectMapper(objectMapper);
        httpMessageConverters.add(jsonConverter);
    }
…

}
```

Метод initBinder контроллера InvoiceController описывает, каким образом текстовое представление даты, присылаемое браузером, преобразуется в значение типа Timestamp.

JSP страница содержит разметку для отображения сетки с шапками счёт-фактур и панель навигации. Позиции счёт фактур отображаются при раскрытии счёт шапки фактуры, как выпадающий грид.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="cp" value="${pageContext.request.servletContext.contextPath}" scope="request" />

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Пример Spring MVC приложения с использованием Firebird и jOOQ</title>

        <!-- Скрипты и стили -->
        <%@ include file="../jspf/head.jspf" %>
        <script src="${cp}/resources/js/jqGridProduct.js"></script>
        <script src="${cp}/resources/js/jqGridCustomer.js"></script>
        <script src="${cp}/resources/js/jqGridInvoice.js"></script>
    </head>
    <body>
        <!-- Навигационное меню -->
        <%@ include file="../jspf/menu.jspf" %>

        <div class="container body-content">

            <h2>Invoices</h2>

            <table id="jqGridInvoice"></table>
            <div id="jqPagerInvoice"></div>

            <hr />
            <footer>
                <p>&copy; 2016 - Пример Spring MVC приложения с использованием Firebird и
jOOQ</p>
            </footer>
        </div>

        <script type="text/javascript">
            var invoiceGrid = null;
            $(document).ready(function () {
                invoiceGrid = JqGridInvoice({
                    baseAddress: '${cp}'
                });
            });
        </script>

    </body>
</html>
```

Основная логика на стороне клиента сосредоточена в JavaScript модуле */resources/js/jqGridInvoice.js*

```
var JqGridInvoice = (function ($, jqGridProductFactory, jqGridCustomerFactory) {

    return function (options) {
        var jqGridInvoice = {
            dbGrid: null,
            detailGrid: null,
            // опции
            options: $.extend({
                baseAddress: null
            }, options),
            // возвращает опции колонок (модель) счёт фактуры
            getInvoiceColModel: function () {
                return [
                    {
```

```
        label: 'Id', // подпись
        name: 'INVOICE_ID', // имя поля
        key: true, // признак ключевого поля
        hidden: true // скрыт
    },
    {
        label: 'Customer Id', // подпись
        name: 'CUSTOMER_ID', // имя поля
        hidden: true, // скрыт
        editrules: {edithidden: true, required: true}, // скрытое и требуемое
        editable: true, // редактируемое
        edittype: 'custom', // собственный тип
        editoptions: {
            custom_element: function (value, options) {
                // добавляем скрытый input
                return $("<input>")
                        .attr('type', 'hidden')
                        .attr('rowid', options.rowId)
                        .addClass("FormElement")
                        .addClass("form-control")
                        .val(value)
                        .get(0);
            }
        }
    },
    {
        label: 'Date',
        name: 'INVOICE_DATE',
        width: 60, // ширина
        sortable: true, // позволять сортировку
        editable: true, // редактируемое
        search: true, // разрешён поиск
        edittype: "text", // тип поля ввода
        align: "right", // выравнено по правому краю
        formatter: jqGridInvoice.dateTimeFormatter, // отформатировано как дата
        sorttype: 'date', // сортируем как дату
        formatoptions: {
            srcformat: 'Y-m-d\TH:i:s', // входной формат
            newformat: 'd.m.Y H:i:s'   // выходной формат
        },
        editoptions: {
            // инициализация элемента формы для редактирования
            dataInit: function (element) {
                // создаём datepicker
                $(element).datepicker({
                    id: 'invoiceDate_datePicker',
                    dateFormat: 'dd.mm.yy',
                    minDate: new Date(2000, 0, 1),
                    maxDate: new Date(2030, 0, 1)
                });
            }
        },
        searchoptions: {
            // инициализация элемента формы для поиска
            dataInit: function (element) {
                // создаём datepicker
                $(element).datepicker({
                    id: 'invoiceDate_datePicker',
                    dateFormat: 'dd.mm.yy',
                    minDate: new Date(2000, 0, 1),
                    maxDate: new Date(2030, 0, 1)
                });
            },
            searchoptions: {// типы поиска
                sopt: ['eq', 'lt', 'le', 'gt', 'ge']
            }
        }
    },
    {
        label: 'Customer',
        name: 'CUSTOMER_NAME',
```

```
                width: 250,
                editable: true,
                edittype: "text",
                editoptions: {
                    size: 50,
                    maxlength: 60,
                    readonly: true    // только чтение
                },
                editrules: {required: true},
                search: true,
                searchoptions: {
                    sopt: ['eq', 'bw', 'cn']
                }
            },
            {
                label: 'Amount',
                name: 'TOTAL_SALE',
                width: 60,
                sortable: false,
                editable: false,
                search: false,
                align: "right",
                formatter: 'currency', // форматировать как валюту
                sorttype: 'number',
                searchrules: {
                    "required": true,
                    "number": true,
                    "minValue": 0
                }
            },
            {
                label: 'Paid',
                name: 'PAID',
                width: 30,
                sortable: false,
                editable: true,
                search: true,
                searchoptions: {
                    sopt: ['eq']
                },
                edittype: "checkbox", // галочка
                formatter: "checkbox",
                stype: "checkbox",
                align: "center",
                editoptions: {
                    value: "1",
                    offval: "0"
                }
            }
        ];
    },
    initGrid: function () {
        // url для получения данных
        var url = jqGridInvoice.options.baseAddress + '/invoice/getdata';
        jqGridInvoice.dbGrid = $("#jqGridInvoice").jqGrid({
            url: url,
            datatype: "json", // формат получения данных
            mtype: "GET", // тип http запроса
            // описание модели
            colModel: jqGridInvoice.getInvoiceColModel(),
            rowNum: 500, // число отображаемых строк
            loadonce: false, // загрузка только один раз
            sortname: 'INVOICE_DATE', // сортировка по умолчанию по столбцу даты
            sortorder: "desc", // порядок сортировки
            width: window.innerWidth - 80, // ширина грида
            height: 500, // высота грида
            viewrecords: true, // отображать количество записей
            guiStyle: "bootstrap",
            iconSet: "fontAwesome",
            caption: "Invoices", // подпись к гриду
            pager: '#jqPagerInvoice', // элемент для отображения постраничной навигации
            subGrid: true, // показывать вложенвй грид
```

```
                    // javascript функция для отображения родительского грида
                    subGridRowExpanded: jqGridInvoice.showChildGrid,
                    subGridOptions: { // опции вложенного грида
                        // загружать данные только один раз
                        reloadOnExpand: false,
                        // загружать строки подгрида только при щелчке по иконке "+"
                        selectOnExpand: true
                    }
                });
            },
            // функция форматирования даты
            dateTimeFormatter: function(cellvalue, options, rowObject) {
                var date = new Date(cellvalue);
                return date.toLocaleString().replace(",", "");
            },
            // возвращает шаблон диалога редактирования
            getTemplate: function () {
                var template = "<div style='margin-left:15px;' id='dlgEditInvoice'>";
                template += "<div>{CUSTOMER_ID} </div>";
                template += "<div> Date: </div><div>{INVOICE_DATE}</div>";
                // поле ввода заказчика с кнопкой
                template += "<div> Customer <sup>*</sup>:</div>";
                template += "<div>";
                template += "<div style='float: left;'>{CUSTOMER_NAME}</div> ";
                template += "<a style='margin-left: 0.2em;' class='btn'
onclick='invoiceGrid.showCustomerWindow(); return false;'>";
                template += "<span class='glyphicon glyphicon-folder-open'></span>Выбрать</a> ";
                template += "<div style='clear: both;'></div>";
                template += "</div>";
                template += "<div> {PAID} Paid </div>";
                template += "<hr style='width: 100%;'/>";
                template += "<div> {sData} {cData}  </div>";
                template += "</div>";
                return template;
            },
            // преобразование даты в UTC
            convertToUTC: function(datetime) {
                if (datetime) {
                    // дату надо преобразовать
                    var dateParts = datetime.split('.');
                    var date = dateParts[2].substring(0, 4) + '-' + dateParts[1] + '-' +
dateParts[0];
                    var time = dateParts[2].substring(5);
                    if (!time) {
                        time = '00:00:00';
                    }
                    var dt = Date.parse(date + 'T' + time);
                    var s = dt.getUTCFullYear() + '-' +
                            dt.getUTCMonth() + '-' +
                            dt.getUTCDay() + 'T' +
                            dt.getUTCHour() + ':' +
                            dt.getUTCMinute() + ':' +
                            dt.getUTCSecond() + '  GMT';
                    return s;
                } else
                    return null;
            },
            // возвращает опции редактирования счёт-фактуры
            getEditInvoiceOptions: function () {
                return {
                    url: jqGridInvoice.options.baseAddress + '/invoice/edit',
                    reloadAfterSubmit: true,
                    closeOnEscape: true,
                    closeAfterEdit: true,
                    drag: true,
                    modal: true,
                    top: $(".container.body-content").position().top + 150,
                    left: $(".container.body-content").position().left + 150,
                    template: jqGridInvoice.getTemplate(),
                    afterSubmit: jqGridInvoice.afterSubmit,
                    editData: {
                        INVOICE_ID: function () {
```

```
                var selectedRow = jqGridInvoice.dbGrid.getGridParam("selrow");
                var value = jqGridInvoice.dbGrid.getCell(selectedRow, 'INVOICE_ID');
                return value;
            },
            CUSTOMER_ID: function () {
                return $('#dlgEditInvoice input[name=CUSTOMER_ID]').val();
            },
            INVOICE_DATE: function () {
                var datetime = $('#dlgEditInvoice input[name=INVOICE_DATE]').val();
                return jqGridInvoice.convertToUTC(datetime);
            }
        }
    };
},
// возвращает опции добавления счёт-фактуры
getAddInvoiceOptions: function () {
    return {
        url: jqGridInvoice.options.baseAddress + '/invoice/create',
        reloadAfterSubmit: true,
        closeOnEscape: true,
        closeAfterAdd: true,
        drag: true,
        modal: true,
        top: $(".container.body-content").position().top + 150,
        left: $(".container.body-content").position().left + 150,
        template: jqGridInvoice.getTemplate(),
        afterSubmit: jqGridInvoice.afterSubmit,
        editData: {
            CUSTOMER_ID: function () {
                return $('#dlgEditInvoice input[name=CUSTOMER_ID]').val();
            },
            INVOICE_DATE: function () {
                var datetime = $('#dlgEditInvoice input[name=INVOICE_DATE]').val();
                return jqGridInvoice.convertToUTC(datetime);
            }
        }
    };
},
// возвращает опции редактирования счёт-фактуры
getDeleteInvoiceOptions: function () {
    return {
        url: jqGridInvoice.options.baseAddress + '/invoice/delete',
        reloadAfterSubmit: true,
        closeOnEscape: true,
        closeAfterDelete: true,
        drag: true,
        msg: "Удалить выделенную счёт-фактуру?",
        afterSubmit: jqGridInvoice.afterSubmit,
        delData: {
            INVOICE_ID: function () {
                var selectedRow = jqGridInvoice.dbGrid.getGridParam("selrow");
                var value = jqGridInvoice.dbGrid.getCell(selectedRow, 'INVOICE_ID');
                return value;
            }
        }
    };
},
initPager: function () {
    // отображение панели навигации
    jqGridInvoice.dbGrid.jqGrid('navGrid', '#jqPagerInvoice',
            {
                search: true, // поиск
                add: true, // добавление
                edit: true, // редактирование
                del: true, // удаление
                view: false, // просмотр записи
                refresh: true, // обновление

                searchtext: "Поиск",
                addtext: "Добавить",
                edittext: "Изменить",
                deltext: "Удалить",
```

```
                    viewtext: "Смотреть",
                    viewtitle: "Выбранная запись",
                    refreshtext: "Обновить"
                },
                jqGridInvoice.getEditInvoiceOptions(),
                jqGridInvoice.getAddInvoiceOptions(),
                jqGridInvoice.getDeleteInvoiceOptions()
                );
        // добавление кнопки для оплаты счёт фактуры
        var urlPay = jqGridInvoice.options.baseAddress + '/invoice/pay';
        jqGridInvoice.dbGrid.navButtonAdd('#jqPagerInvoice',
                {
                    buttonicon: "glyphicon-usd",
                    title: "Оплатить",
                    caption: "Оплатить",
                    position: "last",
                    onClickButton: function () {
                        // получаем идентификатор текущей записи
                        var id = jqGridInvoice.dbGrid.getGridParam("selrow");
                        if (id) {
                            $.ajax({
                                url: urlPay,
                                type: 'POST',
                                data: {INVOICE_ID: id},
                                success: function (data) {
                                    // проверяем не произошла ли ошибка
                                    if (data.hasOwnProperty("error")) {
                                        jqGridInvoice.alertDialog('Ошибка', data.error);
                                    } else {
                                        // обновление грида
                                        $("#jqGridInvoice").jqGrid(
                                                'setGridParam',
                                                {
                                                    datatype: 'json'
                                                }
                                        ).trigger('reloadGrid');
                                    }
                                }
                            });
                        }
                    }
                }
        );
    },
    init: function () {
        jqGridInvoice.initGrid();
        jqGridInvoice.initPager();
    },
    afterSubmit: function (response, postdata) {
        var responseData = response.responseJSON;
        // проверяем результат на наличие сообщений об ошибках
        if (responseData.hasOwnProperty("error")) {
            if (responseData.error.length) {
                return [false, responseData.error];
            }
        } else {
            // обновление грида
            $(this).jqGrid(
                    'setGridParam',
                    {
                        datatype: 'json'
                    }
            ).trigger('reloadGrid');
        }
        return [true, "", 0];
    },
    getInvoiceLineColModel: function (parentRowKey) {
        return [
            {
                label: 'Invoice Line ID',
                name: 'INVOICE_LINE_ID',
                key: true,
```

```
        hidden: true
    },
    {
        label: 'Invoice ID',
        name: 'INVOICE_ID',
        hidden: true,
        editrules: {edithidden: true, required: true},
        editable: true,
        edittype: 'custom',
        editoptions: {
            custom_element: function (value, options) {
                // создаём скрытый элемент ввода
                return $("<input>")
                        .attr('type', 'hidden')
                        .attr('rowid', options.rowId)
                        .addClass("FormElement")
                        .addClass("form-control")
                        .val(parentRowKey)
                        .get(0);
            }
        }
    },
    {
        label: 'Product ID',
        name: 'PRODUCT_ID',
        hidden: true,
        editrules: {edithidden: true, required: true},
        editable: true,
        edittype: 'custom',
        editoptions: {
            custom_element: function (value, options) {
                // создаём скрытый элемент ввода
                return $("<input>")
                        .attr('type', 'hidden')
                        .attr('rowid', options.rowId)
                        .addClass("FormElement")
                        .addClass("form-control")
                        .val(value)
                        .get(0);
            }
        }
    },
    {
        label: 'Product',
        name: 'PRODUCT_NAME',
        width: 300,
        editable: true,
        edittype: "text",
        editoptions: {
            size: 50,
            maxlength: 60,
            readonly: true
        },
        editrules: {required: true}
    },
    {
        label: 'Price',
        name: 'SALE_PRICE',
        formatter: 'currency',
        editable: true,
        editoptions: {
            readonly: true
        },
        align: "right",
        width: 100
    },
    {
        label: 'Quantity',
        name: 'QUANTITY',
        align: "right",
        width: 100,
        editable: true,
```

```
                        editrules: {required: true, number: true, minValue: 1},
                        editoptions: {
                            dataEvents: [
                                {
                                    type: 'change',
                                    fn: function (e) {
                                        var quantity = $(this).val() - 0;
                                        var price = $('#dlgEditInvoiceLine
input[name=SALE_PRICE]').val() - 0;
                                        $('#dlgEditInvoiceLine input[name=TOTAL]').val(quantity *
price);
                                    }
                                }
                            ],
                            defaultValue: 1
                        }
                    },
                    {
                        label: 'Total',
                        name: 'TOTAL',
                        formatter: 'currency',
                        align: "right",
                        width: 100,
                        editable: true,
                        editoptions: {
                            readonly: true
                        }
                    }
                ];
            },
            // возвращает опции редактирования позиции счёт фактуры
            getEditInvoiceLineOptions: function () {
                return {
                    url: jqGridInvoice.options.baseAddress + '/invoice/editdetail',
                    reloadAfterSubmit: true,
                    closeOnEscape: true,
                    closeAfterEdit: true,
                    drag: true,
                    modal: true,
                    top: $(".container.body-content").position().top + 150,
                    left: $(".container.body-content").position().left + 150,
                    template: jqGridInvoice.getTemplateDetail(),
                    afterSubmit: jqGridInvoice.afterSubmit,
                    editData: {
                        INVOICE_LINE_ID: function () {
                            var selectedRow = jqGridInvoice.detailGrid.getGridParam("selrow");
                            var value = jqGridInvoice.detailGrid.getCell(selectedRow,
'INVOICE_LINE_ID');
                            return value;
                        },
                        QUANTITY: function () {
                            return $('#dlgEditInvoiceLine input[name=QUANTITY]').val();
                        }
                    }
                };
            },
            // возвращает опции добавления позиции счёт фактуры
            getAddInvoiceLineOptions: function () {
                return {
                    url: jqGridInvoice.options.baseAddress + '/invoice/createdetail',
                    reloadAfterSubmit: true,
                    closeOnEscape: true,
                    closeAfterAdd: true,
                    drag: true,
                    modal: true,
                    top: $(".container.body-content").position().top + 150,
                    left: $(".container.body-content").position().left + 150,
                    template: jqGridInvoice.getTemplateDetail(),
                    afterSubmit: jqGridInvoice.afterSubmit,
                    editData: {
                        INVOICE_ID: function () {
                            var selectedRow = jqGridInvoice.dbGrid.getGridParam("selrow");
```

```
                            var value = jqGridInvoice.dbGrid.getCell(selectedRow, 'INVOICE_ID');
                            return value;
                        },
                        PRODUCT_ID: function () {
                            return $('#dlgEditInvoiceLine input[name=PRODUCT_ID]').val();
                        },
                        QUANTITY: function () {
                            return $('#dlgEditInvoiceLine input[name=QUANTITY]').val();
                        }
                    }
                };
            },
            // возвращает опции удаления позиции счёт фактуры
            getDeleteInvoiceLineOptions: function () {
                return {
                    url: jqGridInvoice.options.baseAddress + '/invoice/deletedetail',
                    reloadAfterSubmit: true,
                    closeOnEscape: true,
                    closeAfterDelete: true,
                    drag: true,
                    msg: "Удалить выделенную позицию?",
                    afterSubmit: jqGridInvoice.afterSubmit,
                    delData: {
                        INVOICE_LINE_ID: function () {
                            var selectedRow = jqGridInvoice.detailGrid.getGridParam("selrow");
                            var value = jqGridInvoice.detailGrid.getCell(selectedRow,
'INVOICE_LINE_ID');
                            return value;
                        }
                    }
                };
            },
            // обработчик события раскрытия родительского грида
            // принимает два параметра идентификатор родительской записи
            // и первичный ключ записи
            showChildGrid: function (parentRowID, parentRowKey) {
                var childGridID = parentRowID + "_table";
                var childGridPagerID = parentRowID + "_pager";
                // отправляем первичный ключ родительской записи
                // чтобы отфильтровать записи позиций накладной
                var childGridURL = jqGridInvoice.options.baseAddress + '/invoice/getdetaildata';
                childGridURL = childGridURL + "?INVOICE_ID=" + encodeURIComponent(parentRowKey);
                // добавляем HTML элементы для отображения таблицы и постраничной навигации
                // как дочерние для выбранной строки в мастер гриде
                $('<table>')
                        .attr('id', childGridID)
                        .appendTo($('#' + parentRowID));
                $('<div>')
                        .attr('id', childGridPagerID)
                        .addClass('scroll')
                        .appendTo($('#' + parentRowID));
                // создаём и инициализируем дочерний грид
                jqGridInvoice.detailGrid = $("#" + childGridID).jqGrid({
                    url: childGridURL,
                    mtype: "GET",
                    datatype: "json",
                    page: 1,
                    colModel: jqGridInvoice.getInvoiceLineColModel(parentRowKey),
                    loadonce: false,
                    width: '100%',
                    height: '100%',
                    guiStyle: "bootstrap",
                    iconSet: "fontAwesome",
                    pager: "#" + childGridPagerID
                });
                // отображение панели инструментов
                $("#" + childGridID).jqGrid('navGrid', '#' + childGridPagerID,
                        {
                            search: false, // поиск
                            add: true, // добавление
                            edit: true, // редактирование
                            del: true, // удаление
```

```
                            refresh: true // обновление
                        },
                        jqGridInvoice.getEditInvoiceLineOptions(),
                        jqGridInvoice.getAddInvoiceLineOptions(),
                        jqGridInvoice.getDeleteInvoiceLineOptions()
                        );
            },
            // возвращает шаблон для редактора позиции счёт фактуры
            getTemplateDetail: function () {
                var template = "<div style='margin-left:15px;' id='dlgEditInvoiceLine'>";
                template += "<div>{INVOICE_ID} </div>";
                template += "<div>{PRODUCT_ID} </div>";
                // поле ввода товара с кнопкой
                template += "<div> Product <sup>*</sup>:</div>";
                template += "<div>";
                template += "<div style='float: left;'>{PRODUCT_NAME}</div> ";
                template += "<a style='margin-left: 0.2em;' class='btn'
onclick='invoiceGrid.showProductWindow(); return false;'>";
                template += "<span class='glyphicon glyphicon-folder-open'></span> Выбрать</a> ";
                template += "<div style='clear: both;'></div>";
                template += "</div>";
                template += "<div> Quantity: </div><div>{QUANTITY} </div>";
                template += "<div> Price: </div><div>{SALE_PRICE} </div>";
                template += "<div> Total: </div><div>{TOTAL} </div>";
                template += "<hr style='width: 100%;'/>";
                template += "<div> {sData} {cData}  </div>";
                template += "</div>";
                return template;
            },
            // отображение окна выбора продукта из справочника
            showProductWindow: function () {
                var dlg = $('<div>')
                        .attr('id', 'dlgChooseProduct')
                        .attr('aria-hidden', 'true')
                        .attr('role', 'dialog')
                        .attr('data-backdrop', 'static')
                        .css("z-index", '2000')
                        .addClass('modal')
                        .appendTo($('body'));

                var dlgContent = $("<div>")
                        .addClass("modal-content")
                        .css('width', '760px')
                        .appendTo($('<div>')
                                .addClass('modal-dialog')
                                .appendTo(dlg));

                var dlgHeader = $('<div>').addClass("modal-header").appendTo(dlgContent);
                $("<button>")
                        .addClass("close")
                        .attr('type', 'button')
                        .attr('aria-hidden', 'true')
                        .attr('data-dismiss', 'modal')
                        .html("&times;")
                        .appendTo(dlgHeader);
                $("<h5>").addClass("modal-title").html("Выбор заказчика").appendTo(dlgHeader);
                var dlgBody = $('<div>')
                        .addClass("modal-body")
                        .appendTo(dlgContent);
                var dlgFooter = $('<div>').addClass("modal-footer").appendTo(dlgContent);
                $("<button>")
                        .attr('type', 'button')
                        .addClass('btn')
                        .html('OK')
                        .on('click', function () {
                            var rowId = $("#jqGridProduct").jqGrid("getGridParam", "selrow");
                            var row = $("#jqGridProduct").jqGrid("getRowData", rowId);
                            $('#dlgEditInvoiceLine
input[name=PRODUCT_ID]').val(row["PRODUCT_ID"]);
                            $('#dlgEditInvoiceLine input[name=PRODUCT_NAME]').val(row["NAME"]);
                            $('#dlgEditInvoiceLine input[name=SALE_PRICE]').val(row["PRICE"]);
```

```
                            var price = $('#dlgEditInvoiceLine input[name=SALE_PRICE]').val() -
0;
                            var quantity = $('#dlgEditInvoiceLine input[name=QUANTITY]').val() -
0;
                            $('#dlgEditInvoiceLine input[name=TOTAL]').val(Math.round(price *
quantity * 100) / 100);
                            dlg.modal('hide');
                        })
                        .appendTo(dlgFooter);

                $("<button>")
                        .attr('type', 'button')
                        .addClass('btn')
                        .html('Cancel')
                        .on('click', function () {
                            dlg.modal('hide');
                        })
                        .appendTo(dlgFooter);

                $('<table>')
                        .attr('id', 'jqGridProduct')
                        .appendTo(dlgBody);
                $('<div>')
                        .attr('id', 'jqPagerProduct')
                        .appendTo(dlgBody);

                dlg.on('hidden.bs.modal', function () {
                    dlg.remove();
                });
                dlg.modal();

                jqGridProductFactory({
                    baseAddress: jqGridInvoice.options.baseAddress
                });
            },
            // отображение окна выбора заказчика из справочника
            showCustomerWindow: function () {
                // основной блок диалога
                var dlg = $('<div>')
                        .attr('id', 'dlgChooseCustomer')
                        .attr('aria-hidden', 'true')
                        .attr('role', 'dialog')
                        .attr('data-backdrop', 'static')
                        .css("z-index", '2000')
                        .addClass('modal')
                        .appendTo($('body'));
                // блок с содержимым диалога
                var dlgContent = $("<div>")
                        .addClass("modal-content")
                        .css('width', '730px')
                        .appendTo($('<div>')
                                .addClass('modal-dialog')
                                .appendTo(dlg));
                // блок с шапкой диалога
                var dlgHeader = $('<div>').addClass("modal-header").appendTo(dlgContent);
                // кнопка "X" для закрытия
                $("<button>")
                        .addClass("close")
                        .attr('type', 'button')
                        .attr('aria-hidden', 'true')
                        .attr('data-dismiss', 'modal')
                        .html("&times;")
                        .appendTo(dlgHeader);
                // подпись
                $("<h5>").addClass("modal-title").html("Выбор заказчика").appendTo(dlgHeader);
                // тело диалога
                var dlgBody = $('<div>')
                        .addClass("modal-body")
                        .appendTo(dlgContent);
                // подвал диалога
                var dlgFooter = $('<div>').addClass("modal-footer").appendTo(dlgContent);
                // Кнопка "OK"
```

```
            $("<button>")
                    .attr('type', 'button')
                    .addClass('btn')
                    .html('OK')
                    .on('click', function () {
                        var rowId = $("#jqGridCustomer").jqGrid("getGridParam", "selrow");
                        var row = $("#jqGridCustomer").jqGrid("getRowData", rowId);
                        // сохраняем идентификатор и имя заказчика
                        // в элементы ввода родительской формы
                        $('#dlgEditInvoice input[name=CUSTOMER_ID]').val(rowId);
                        $('#dlgEditInvoice input[name=CUSTOMER_NAME]').val(row["NAME"]);
                        dlg.modal('hide');
                    })
                    .appendTo(dlgFooter);
            // Кнопка "Cancel"
            $("<button>")
                    .attr('type', 'button')
                    .addClass('btn')
                    .html('Cancel')
                    .on('click', function () {
                        dlg.modal('hide');
                    })
                    .appendTo(dlgFooter);
            // добавляем таблицу для отображения заказчиков в тело диалога
            $('<table>')
                    .attr('id', 'jqGridCustomer')
                    .appendTo(dlgBody);
            // добавляем панель навигации
            $('<div>')
                    .attr('id', 'jqPagerCustomer')
                    .appendTo(dlgBody);
            dlg.on('hidden.bs.modal', function () {
                dlg.remove();
            });
            // отображаем диалог
            dlg.modal();
            jqGridCustomerFactory({
                baseAddress: jqGridInvoice.options.baseAddress
            });
        },
        // Окно для отображения ошибки
        alertDialog: function (title, error) {
            var alertDlg = $('<div>')
                    .attr('aria-hidden', 'true')
                    .attr('role', 'dialog')
                    .attr('data-backdrop', 'static')
                    .addClass('modal')
                    .appendTo($('body'));
            var dlgContent = $("<div>")
                    .addClass("modal-content")
                    .appendTo($('<div>')
                            .addClass('modal-dialog')
                            .appendTo(alertDlg));
            var dlgHeader = $('<div>').addClass("modal-header").appendTo(dlgContent);
            $("<button>")
                    .addClass("close")
                    .attr('type', 'button')
                    .attr('aria-hidden', 'true')
                    .attr('data-dismiss', 'modal')
                    .html("&times;")
                    .appendTo(dlgHeader);
            $("<h5>").addClass("modal-title").html(title).appendTo(dlgHeader);
            $('<div>')
                    .addClass("modal-body")
                    .appendTo(dlgContent)
                    .append(error);
            alertDlg.on('hidden.bs.modal', function () {
                alertDlg.remove();
            });
            alertDlg.modal();
        }
    };
```

```
        jqGridInvoice.init();
        return jqGridInvoice;
    };
})(jQuery, JqGridProduct, JqGridCustomer);
```

В журнале счёт фактур, основная сетка используется для отображения шапок, а раскрывающаяся по клику для отображения позиций. Для отображения дочернего грида свойству `subGrid` присвоено значение true. Дочерняя сетка отображается, используя событие `subGridRowExpanded`, которое связано с методом `showChildGrid`. Позиции фильтруются по первичному ключу счёт фактуры. Помимо основных кнопок панель навигации для шапки счёт фактуры добавлена пользовательская кнопка для оплаты счёт фактуры с помощью функции `jqGridInvoice.dbGrid.navButtonAdd` (см. метод `initPager`).

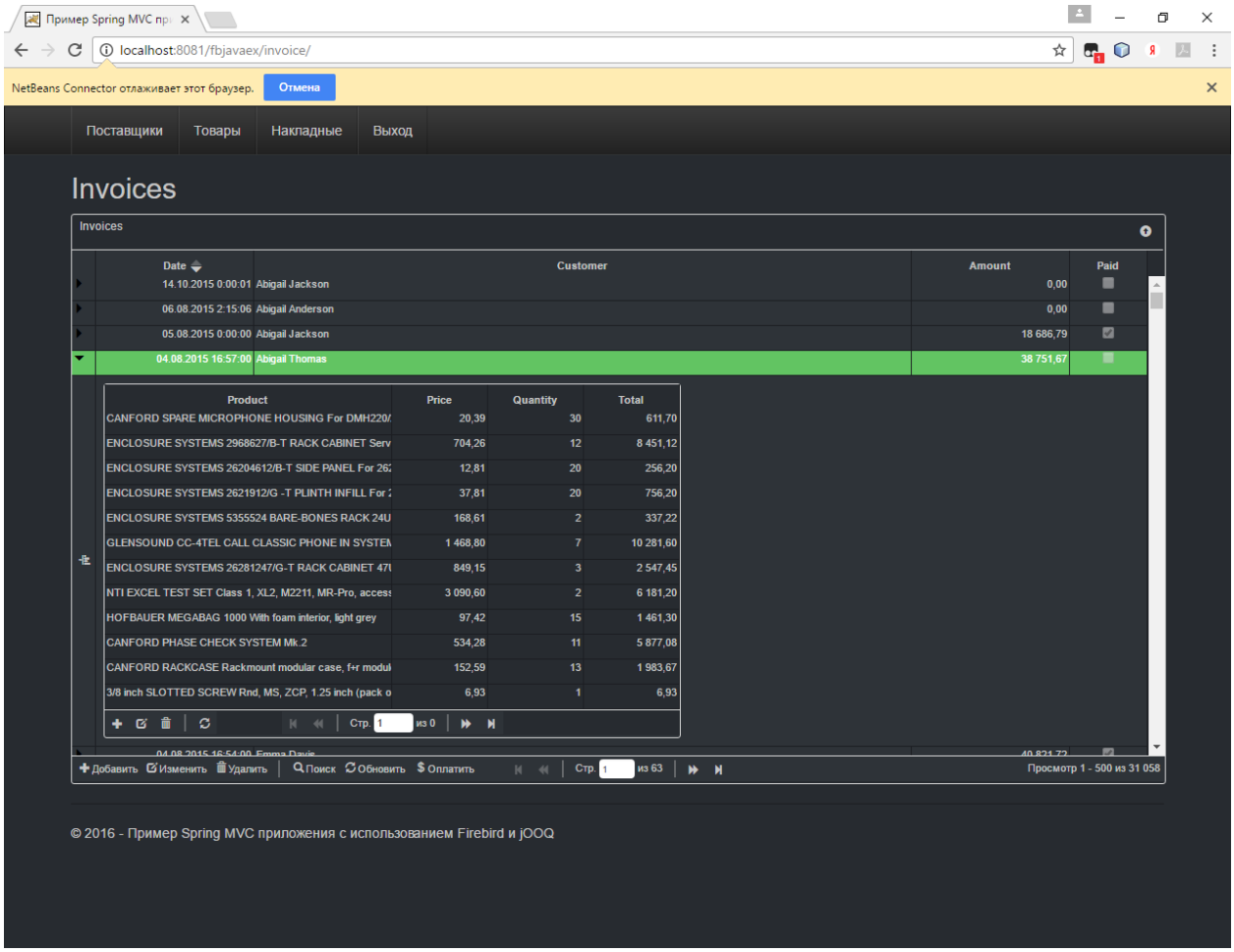
В отличие от справочников диалоги редактирования для журналов намного сложнее. Зачастую они используют выбор из других справочников. Поэтому такие диалоги редактирования не получится построить автоматически с помощью jqGrid, однако в этой библиотеки существует возможность построение диалогов по шаблону, которой мы и воспользуемся. Шаблон диалога возвращает функцией getTemplate. Открытие справочника заказчиков для выбора заказчика осуществляется функцией invoiceGrid.showCustomerWindow(). Она использует функции уже описанного ранее модуля JqGridCustomer. После выбора заказчика из модального окна его код подставляется в поле CUSTOMER_ID. В свойстве editData опций редактирования и добавления описаны поля, которые надо будет передать на сервер, используя предварительную обработку или из невидимых полей.

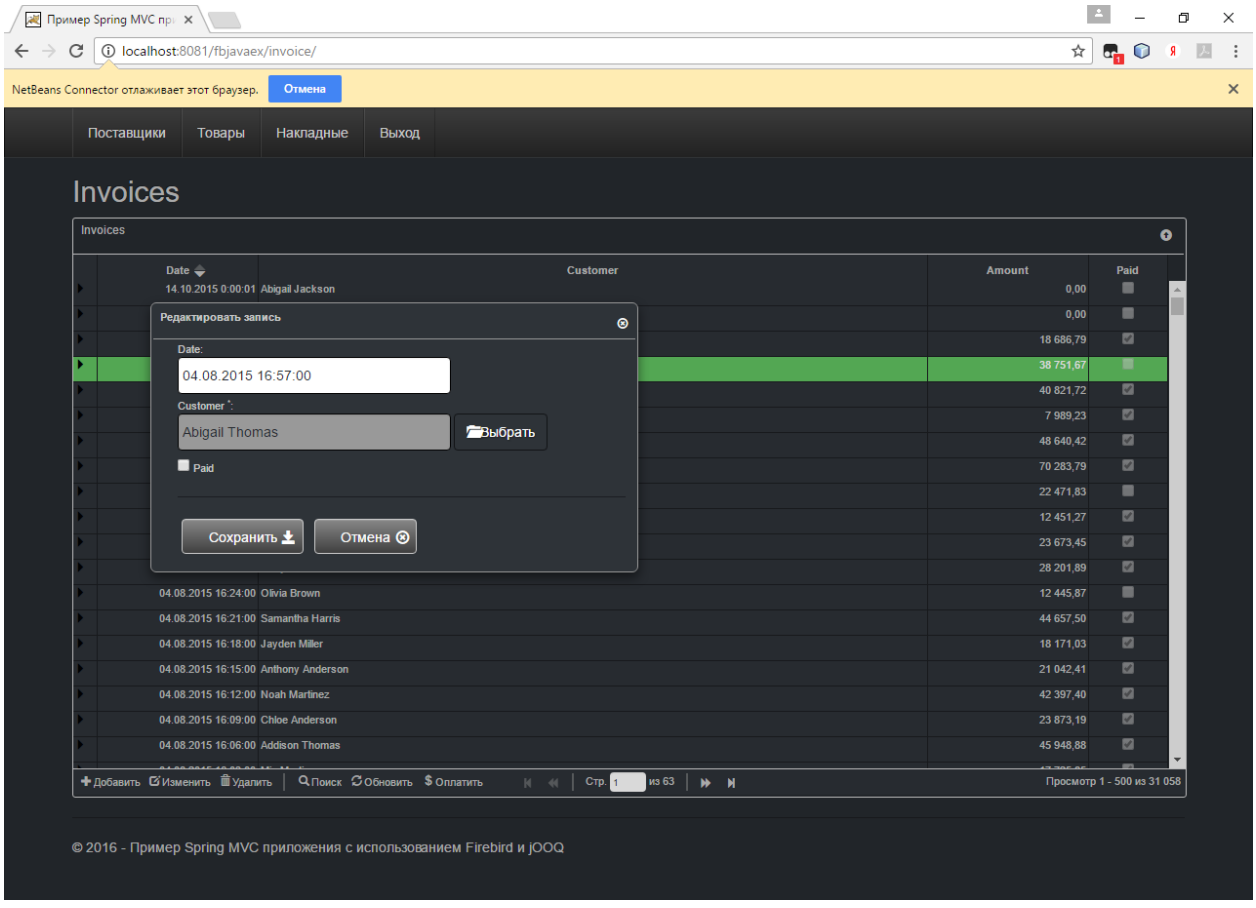
Теперь вернёмся к обработке дат. Как я уже говорил, контроллер InvoiceController возвращает дату в UTC, нам же необходимо отобразить её в текущей часовой зоне. Для этого зададим функцию форматирования даты jqGridInvoice.dateTimeFormatter через свойство formatter, соответствующего поля INVOICE_DATE. При отправке данных на сервер нам необходимо сделать обратную операцию – перевести время из текущей временной зоны в UTC. За это отвечает функция convertToUTC.

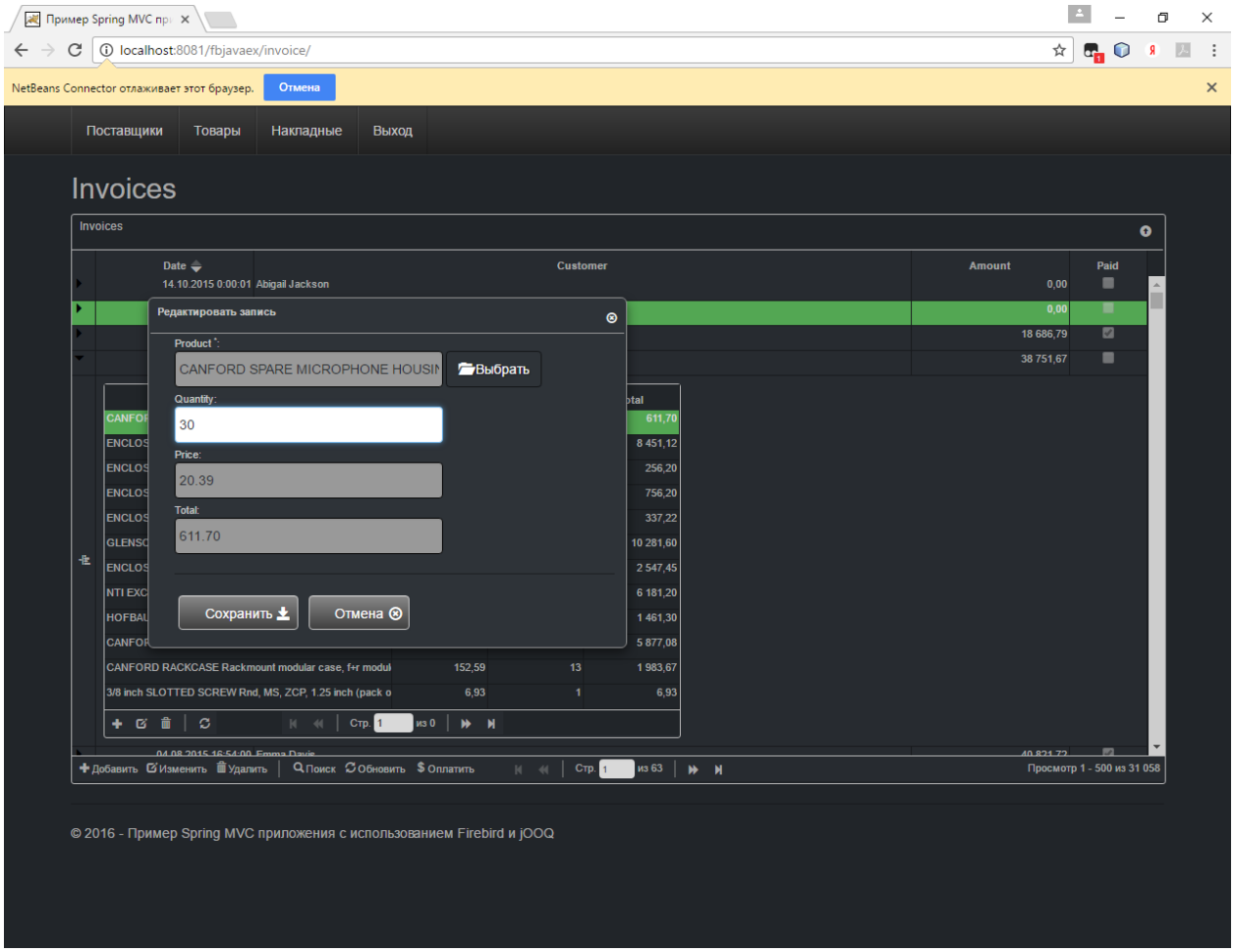
Для редактора позиций счёт фактуры, так же используется пользовательский шаблон, который возвращается функцией getTemplateDetail. Открытие окна для выбора из справочника товаров осуществляется функцией invoiceGrid.showProductWindow(). Эта функция использует функции модуля JqGridProduct.

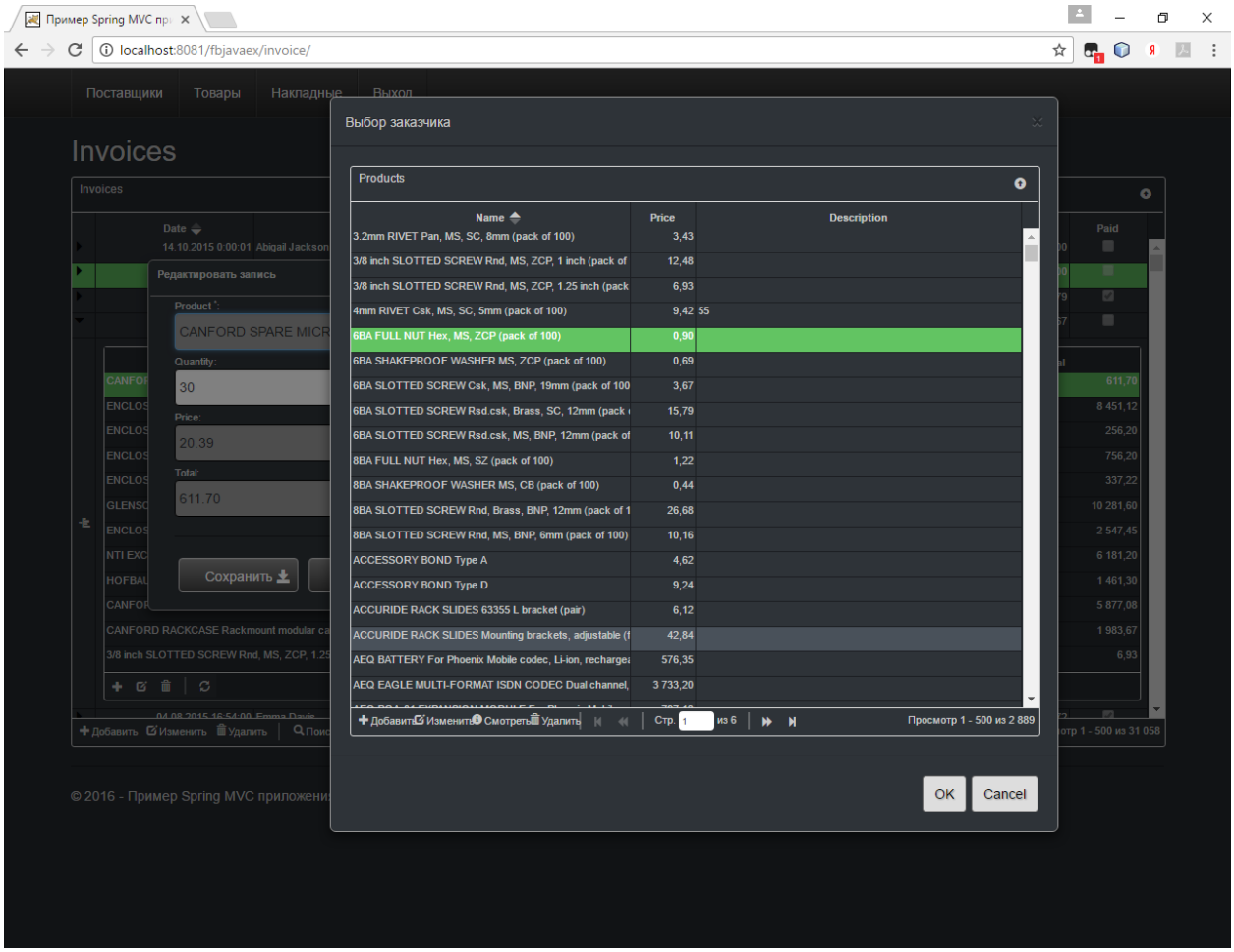
Код модуля JqGridInvoice подробно прокомментирован так, чтобы вы могли понять логику его работы. Дополнительные пояснения вы можете найти в нём.

Напоследок приведу несколько скриншотов получившегося веб приложения.


Пример Spring MVC пр
localhost:8081/fbjavaex/invoice/
NetBeans Connector отлаживает этот браузер.
Отмена
Поставщики
Товары
Накладные
Выход
Invoices
Invoices
Date
Customer
Amount
Paid
14.10.2015 0:00:01 Abigail Jackson 0,00
06.08.2015 2:15:06 Abigail Anderson 0,00
05.08.2015 0:00:00 Abigail Jackson 18 686,79
04.08.2015 16:57:00 Abigail Thomas 38 751,67
Product
Price
Quantity
Total
CANFORD SPARE MICROPHONE HOUSING For DMH220/ 20,39 30 611,70
ENCLOSURE SYSTEMS 2968627/B-T RACK CABINET Serv 704,26 12 8 451,12
ENCLOSURE SYSTEMS 26204612/B-T SIDE PANEL For 26 12,81 20 256,20
ENCLOSURE SYSTEMS 2621912/G -T PLINTH INFILL For 3 37,81 20 756,20
ENCLOSURE SYSTEMS 5355524 BARE-BONES RACK 24U 168,61 2 337,22
GLENSOUND CC-4TEL CALL CLASSIC PHONE IN SYSTEM 1 468,80 7 10 281,60
ENCLOSURE SYSTEMS 26281247/G-T RACK CABINET 47U 849,15 3 2 547,45
NTI EXCEL TEST SET Class 1, XL2, M2211, MR-Pro, access 3 090,60 2 6 181,20
HOFBAUER MEGABAG 1000 With foam interior, light grey 97,42 15 1 461,30
CANFORD PHASE CHECK SYSTEM Mk.2 534,28 11 5 877,08
CANFORD RACKCASE Rackmount modular case, f+r modul 152,59 13 1 983,67
3/8 inch SLOTTED SCREW Rnd, MS, ZCP, 1.25 inch (pack o 6,93 1 6,93
Стр. 1 из 0
Добавить
Изменить
Удалить
Поиск
Обновить
Оплатить
Стр. 1 из 63
Просмотр 1 - 500 из 31 058
© 2016 - Пример Spring MVC приложения с использованием Firebird и jOOQ


Пример Spring MVC пр
localhost:8081/fbjavaex/invoice/
NetBeans Connector отлаживает этот браузер.
Отмена
Поставщики
Товары
Накладные
Выход
Invoices
Invoices
Date
Customer
Amount
Paid
14.10.2015 0:00:01 Abigail Jackson 0,00
Редактировать запись
Date:
04.08.2015 16:57:00
Customer *:
Abigail Thomas
Выбрать
Paid
Сохранить
Отмена
0,00
18 686,79
38 751,67
40 821,72
7 989,23
48 640,42
70 283,79
22 471,83
12 451,27
23 673,45
28 201,89
04.08.2015 16:24:00 Olivia Brown 12 445,87
04.08.2015 16:21:00 Samantha Harris 44 657,50
04.08.2015 16:18:00 Jayden Miller 18 171,03
04.08.2015 16:15:00 Anthony Anderson 21 042,41
04.08.2015 16:12:00 Noah Martinez 42 397,40
04.08.2015 16:09:00 Chloe Anderson 23 873,19
04.08.2015 16:06:00 Addison Thomas 45 948,88
Добавить
Изменить
Удалить
Поиск
Обновить
Оплатить
Стр. 1 из 63
Просмотр 1 - 500 из 31 058
© 2016 - Пример Spring MVC приложения с использованием Firebird и jOOQ


Пример Spring MVC пр
localhost:8081/fbjavaex/invoice/
NetBeans Connector отлаживает этот браузер.
Отмена
Поставщики
Товары
Накладные
Выход
Invoices
Invoices
Date
Customer
Amount
Paid
14.10.2015 0:00:01 Abigail Jackson
0,00
0,00
18 686,79
38 751,67
Редактировать запись
Product *:
CANFORD SPARE MICROPHONE HOUSIN
Выбрать
Quantity:
30
Price:
20.39
Total:
611.70
Сохранить
Отмена
611,70
8 451,12
256,20
756,20
337,22
10 281,60
2 547,45
6 181,20
1 461,30
5 877,08
CANFORD RACKCASE Rackmount modular case, f+r modul 152,59 13 1 983,67
3/8 inch SLOTTED SCREW Rnd, MS, ZCP, 1.25 inch (pack o 6,93 1 6,93
Стр. 1 из 0
Стр. 1 из 63
Добавить
Изменить
Удалить
Поиск
Обновить
$ Оплатить
Просмотр 1 - 500 из 31 058
© 2016 - Пример Spring MVC приложения с использованием Firebird и jOOQ



На этом мой пример закончен. Исходные коды вы можете скачать по ссылке https://github.com/sim1984/phpfbexample.