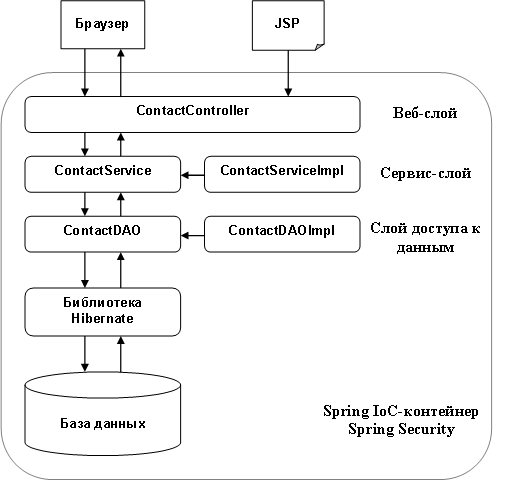
Требуется реализовать прототип магазина.

Приложение должно соответствовать принципам MVC и использовать технологии:, Hibernate, Spring, Spring MVC, Spring Security,JSP.



Пример архитектуры приложения

Для запуска приложения необходимо использовать сервер приложений Tomcat, БД hsqldb. Начальные данные могут быть загружены любым удобным способом (CSV, SQL скрипты).

Приложение должно содержать страницы:

Форма логина



Начальная страница авторизованного пользователя



Артикул: Артикул товара

Категория: Ссылка на страницу со списком товаров из одной категории

Название: Название продукта, ссылка на страницу деталей продукта.

Колл.: Общее количество продуктов на складе, уменьшается при совершении покупки.

Цена: Цена за штуку

Купить - Кнопка добавляет 1 единицу товара в корзину, сумма в правом углу должна изменится соответственно.

Сумма покупок: Ссылка на корзину

Страница деталей продукта



Корзина



После нажатия на кнопку заказать происходит обновление остатков, и пользователь переводится на страницу со списком товаров.

Страница оператора Вывод товаров с 0 значение остатков



Слой DAO и сервисов должен конфигурироваться в Spring через xml, слой контроллеров - через аннотации.

Не изменяя код созданных XXXService и XXXDao и не удаляя объявление бина XXXDaoImpl в xml-конфигурации Spring, сделать так, чтобы на главной странице не выводились товары со значением остатка 0.

**Примечание:**

Нужно создать дополнительный ModifiedContactDaoImpl и используя xml-конфигурацию Spring и тег alias, использовать его вместо ContactDaoImpl.

Код должен сопровождаться Junit тестами - покрытие не менее 50%

**Вариант 1**

Предоставить SOAP API для получения списка товаров

Предоставить Rest API для получения списка товаров

**Вариант 2**

Разработать страницу выгрузки товаров csv

**Вариант 3**

Разработать страницу загрузки товаров из csv

Пример кода загрузки начальных данных из CSV

<dependency>

<groupId>net.sf.opencsv</groupId>

<artifactId>opencsv</artifactId>

<version>1.8</version>

</dependency>

**private void** loadData(Resource resource, Class clazz, CrudRepository crudRepository) **throws** RuntimeException {

**try** {

CSVReader csvReader = **new** CSVReader(**new** InputStreamReader(resource.getInputStream()), **';'**);

String[] columnHeaders = **null**;

String[] record;

**while** ((record = csvReader.readNext()) != **null**) {

**if** (columnHeaders == **null**) {

columnHeaders = record;

} **else** {

Object entity = clazz.newInstance();

**for** (**int** i = 0; i < columnHeaders.**length**; i++) {

String propertyName = columnHeaders[i];

String propertyValue = record[i];

**if** (**null** != propertyName) {

**switch** (propertyName) {

**case "currency"**:

BeanUtils.*setProperty*(entity, propertyName, Currency.valueOf(propertyValue));

**break**;

**default**:

BeanUtils.*setProperty*(entity, propertyName, propertyValue);

**break**;

}

}

}

crudRepository.save(entity);

}

}

} **catch** (Exception ex) {

**throw new** RuntimeException(ex);

}

}