МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных

систем и технологий

**КУРСОВОЙ ПРОЕКТ**

по дисциплине:” Модульное тестирование”

на тему: **«**Электронный магазин для мобильных телефонов»

Выполнил**:** ст.гр.

10701118 **Воробей И.А.**

Принял**:** к.т.н. доц. **Попова Ю.Б.**

Минск 2020

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра программного обеспечения информационных

систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по дисциплине:” Модульное тестирование”

на тему: «Электронный магазин для мобильных телефонов»

Исполнитель**:** Воробей И.А.

Руководитель**:** к.т.н. доц. Попова Ю.Б.

Минск 2020

**Оглавление**

[ВВЕДЕНИЕ 5](#_Toc59134471)

[1. Требования к программному продукту 6](#_Toc59134472)

[Таблица – 1.1 Регистрация 6](#_Toc59134473)

[Таблица – 1.2 Аутентификация 7](#_Toc59134474)

[Таблица – 1.3. Header 8](#_Toc59134475)

[Таблица – 1.4 Footer 8](#_Toc59134476)

[Таблица – 1.5 Гость 9](#_Toc59134477)

[Таблица – 1. 6. Аутентифицированный пользователь (как покупатель) 13](#_Toc59134478)

[Таблица 1.7 - Аутентифицированный пользователь (как админ) 14](#_Toc59134479)

[2. Проектирование программных модулей 17](#_Toc59134480)

[3. Реализация программных модулей 18](#_Toc59134481)

[4. Проектирование модульных тестов 23](#_Toc59134482)

[4.1 Построение ориентированных графов МакКейба 23](#_Toc59134483)

[4.2 Формирование проходов 25](#_Toc59134484)

[5. Проектирование программных модулей 29](#_Toc59134485)

[5.1 Реализация автоматизированных скриптов 29](#_Toc59134486)

[Интеграционное тестирование 31](#_Toc59134487)

[5.2 Анализ результатов тестов 33](#_Toc59134488)

[5.3 Отладка тестов 33](#_Toc59134489)

[ЗАКЛЮЧЕНИЕ 37](#_Toc59134490)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 38](#_Toc59134491)

[Приложение А. Листинг исходных кодов 39](#_Toc59134492)

[Приложение Б. Листинг автоматизированных скриптов 45](#_Toc59134493)

# ВВЕДЕНИЕ

Тестирование играет жизненно важную роль в разработке программного обеспечения и является неотъемлемой частью жизненного цикла его разработки, так как:

* повышает надежность, качество и производительность программного обеспечения,
* помогает разработчику проверить, правильно ли работает программное обеспечение, убедиться, что программное обеспечение выполняет то, для чего оно предназначено.
* помогает понять разницу между фактическим и ожидаемым результатом, что обеспечивает качество продукта.

Благодаря тому, что тестирование ПО становится частью программирования, разработчики имеют возможность исправлять ошибки уже на начальной стадии разработки. Это позволяет сократить риск появления дефектов в готовом продукте. Если ошибки найдены на начальном уровне, разработчик может создать надежное программное обеспечение. Таким образом, чем раньше начинается процесс, тем раньше обнаруживаются ошибки и тем ниже стоимость их исправления.

Без надлежащей оценки производимое программное обеспечение может нанести существенный вред как пользователям, так и производителю. Потеря денег и доверия к компании – это лишь небольшая часть последствий. Хорошее и эффективное тестирование программы помогает повысить его безопасность.

Верификация и валидация являются основными целями процесса. По сути, тестирование программного обеспечения помогает не только обнаружить дефекты в программном обеспечении, но и определить его нужность для клиента. Таким образом, качественное и управляемое тестирование делает бизнес успешным и повышает доверие к компании.

# 1. Требования к программному продукту

В данном разделе описаны требования к системе “Электронный магазин для мобильных телефонов”.

Система предназначена для заказа мобильных телефонов через Интернет.

Сайт должен иметь следующие страницы:

1. Register page
2. Login page
3. PLP (Product list page)
4. PDP (Product details page)
5. Cart page
6. Checkout page
7. Order overview page
8. Admin menu
9. Edit products
10. Create product
11. Reviews moderation

В системе предусмотрено три роли пользователей:

* Не аутентифицированный пользователь(гость)
* Аутентифицированный пользователь (как покупатель)
* Аутентифицированный пользователь (как админ)

В соответствии с этими ролями система делится на три части: сторона гостя, сторона покупателя и сторона администратора. Причем, покупатель может делать, все что может гость и администратор может делать все, что может покупатель. Подробный функцонал описан в таблицах 1.1 – 1.7.

Таблица – 1.1 Регистрация

|  |  |
| --- | --- |
| ID | Описание требования |
| R1.1 | При запуске приложения демонстрируется страница PLP. Ссылка на регистрацию есть в хедере страницы и на странице аутентификации. |
| R1.2 | На странице регистрации содержатся следующие элементы:   * Текстовое поле для ввода логина * Текстовое поле для ввода пароля * Текстовое поле для ввода пароля повторно * Кнопка для входа в систему   Примечание! Тип вводимых данных string[3..20] |
| R1.2.1 | Текстовые поля логин, пароль и повтор пароля являются обязательными для заполнения |
| R 1.2.2 | Логин может содержать только латинские буквы, цифры и знак подчёркивания |
| R 1.2.3 | Если пользователь ввёл недопустимое количество символов в поле логина (меньше 3 или больше 20) , то появляется надпись c сообщением «Логин должен содержать от 3 до 20 символов» |
| R1.2.4 | Имя пользователя чувствительно к регистру |
| R 1.2.5 | Пароль может содержать любые символы |
| R 1.2.6 | Если пользователь ввёл недопустимое количество символов в поле пароля или повтора пароля (меньше 3 или больше 20) , то появляется сообщение «Пароль должен содержать от 3 до 20 символов» |
| R 1.2.7 | Если пользователь ввёл логин который уже зарегистрирован и нажал на конпку «Register», то появляется сообщение «Such a user already exists. Try another login.» |
| R 1.2.8 | После нажатия на кнопку «Register» пользователь получает аккаунт и автоматически аутентифицируется. Открывается PLP |

Таблица – 1.2 Аутентификация

|  |  |
| --- | --- |
| ID | Описанее требования |
| R 2.1 | При запуске приложения демонстрируется страница PLP. В хедере страницы есть ссылка на аутентификацию. |
| R 2.2 | На странице аутентификации содержатся следующие элементы:   * Текстовое поле для ввода логина * Текстовое поле для ввода пароля * Кнопка для входа в систему   Примечание! Тип вводимых данных string[3..20] |
| R 2.2.1 | Текстовые поля логин, пароль являются обязательными для заполнения |
| R 2.2.2 | Логин может содержать только латинские буквы, цифры и знак подчёркивания |
| R 2.2.3 | Если пользователь ввёл недопустимое количество символов в поле логина (меньше 3 или больше 20) , то появляется надпись c сообщением «Логин должен содержать от 3 до 20 символов» |
| R 2.2.4 | Имя пользователя чувствительно к регистру |
| R 2.2.5 | Пароль может содержать любые символы |
| R 2.2.6 | Если пользователь ввёл недопустимое количество символов в поле пароля (меньше 3 или больше 20) , то появляется сообщение «Пароль должен содержать от 3 до 20 символов» |
| R 2.2.7 | Если пользователь неверно ввёл или логин или пароль и нажал на кноппку «Войти», то появляется сообщение «There is no user having such login and password.» |
| R 2.2.8 | После удачного ввода логина, пароля и нажатия на кнопку «Login» пользователь перенаправляется либо на главную страницу(если аутентификация была через ссылкy в хедере), либо на страницу на которую он пытался попасть ранее если она перенаправила его на страницу аутентификации. |

Header является общим для всех страниц. Все пользователи видят его одинаково.

Таблица – 1.3. Header

|  |  |
| --- | --- |
| ID | Описание требования |
| R 3.1 | Header содержит:   * Название магазина со ссылкой на PLP * Поле для поиска продукта по названию и кнопку * Mini cart (значёк корзины с числом добавленных продуктов и суммой) * Register reference * Login reference * Logout button |
| R 3.1.1 | Поле поиска может принимать любую строку либо оставаться пустым. При нажатии на кнопку поиска, пользователь перенаправляется на PLP с результатами поиска. Будут показаны продукты, чье название больше всего похоже на то что ввел пользователь. Продукты выводятся в порядке от «максимально похож» до «минимально похож». Если поле было пустым перед нажатием кнопки, будут показаны все продукты. Продукты отображаются в иде таблицы плиток (с лева на право) шириной 3. Если ничего не найдено по данному запросу, то не отобразится ни одна плитка. И выведется сообщение “по вашему запросу продукты не найдены” |
| R 3.1.2 | Mini cart является ссылкой на корзину |
| R 3.1.3 | Если корзина пуста то информация о количестве и сумме не показывается |
| R 3.1.4 | Register reference fetches the user to register page |
| R 3.1.5 | Login reference fetches the user to login page |
| R 3.1.6 | Logout button logouts the user and redirects to PLP |

Footer является общим для всех страниц. Все пользователи видят его одинаково.

Таблица – 1.4 Footer

|  |  |
| --- | --- |
| ID | Описание требования |
| R 4.1 | Footer contains:   * Recently viewed products panel * Admin menu reference |
| R 4.1.1 | На панели недавно просмотренных продуктов отображается от 1 до 3 продуктов. |
| R 4.1.2 | Если пользователь еще ни разу не открывал PDP ни одного из продуктов, то recently viewed panel не показывается. Для каждого продукта есть фото, название и цена. Фото и название являются ссылками на PDP этого продукта. |
| R 4.1.3 | Ссылка на меню админа видна всем, но перейти по ней может только пользователь, аутентифицированный как админ. |
| R 4.1.4 | Если не аутентифицированный пользователь нажмет на ссылку «админ меню», его перенаправит на страницу аутентификации. |
| R 4.1.5 | Если пользователь аутентифицированный как покупатель нажмет на ссылку «админ меню», то будет выведено сообщение «You do not have enough rights to access this page. If you are a moderator, logout and login as moderator.» |

Гость может просматривать, сортировать, искать продукты по названию, фильтровать по брендам, цене, наличию на складе и добавлять их в корзину. Удалять продукты из корзины, изменять количество продуктов в корзине. Гость видит recently viewed products (список продуктов, у которых пользователь недавно открывал PDP).

Таблица – 1.5 Гость

|  |  |
| --- | --- |
| ID | Описание требования |
| R 5.1 | На PLP гость видит:   * Панель фильтрации * Панель сортировок * Список продуктов |
| R 5.1.1 | Панель фильтрации содержит следующие элементы:   * Список брендов в виде checkbox * Поле для ввода минимальной цены * Поле для ввода максимальной цены * Поле для ввода минимального количества в наличии * Кнопка “Filter” |
| R 5.1.1.1 | Пользователь отмечает галочками бренды, которые он хочет получить. Если не отмечен ни один бренд, то фильтрация по брендам не примениться. |
| R 5.1.1.2 | Поля min price and max price принимают неотрицательные значения с плавающей точкой. В случае передачи строки или отрицательного значения, фильтрация по данному полю не применится. |
| R 5.1.1.3 | Поле min stock принимает неотрицательное значение с плавающей точкой. В случае передачи строки или отрицательного значения, фильтрация по данному полю не применится. |
| R 5.1.2 | Панель сортировок содержит следующие элементы:   * Ссылку на сортировку по имени asc * Ссылку на сортировку по имени desc * Ссылку на сортировку по цене asc * Ссылку на сортировку по цене desc |
| R 5.1.2.1 | При нажатии на стрелку будут отображены результаты поиска (R 3.1.1) и фильтрации (R5.1.1) в отсортированном виде. |
| R 5.1.3 | Продукты отображаются в виде плиток, каждая плитка содержит:   * Фото * Название * Цену * Поле ввода количества * Add to cart button |
| R 5.1.3.1 | Фото и название являются ссылками на PDP |
| R 5.1.3.2 | При нажатии на цену продукта, всплывает popup, показывающий историю изменеия цены. При повторном нажатии либо при нажатии на цену другого продукта, попап предыдущего скрывается. |
| R 5.1.3.2 | По дефолту в поле ввода количества стоит 1. Если пользователь нажимает на кнопку, всплывает сообщение «Added to cart successfully» |
| R 5.1.3.3 | Если пользователь вводит не целое число, то всплывает сообщение «Not a number» |
| R 5.1.3.4 | Если пользователь вводит отрицательное число либо превышающие количество продуктов на складе, то всплывает сообщение «Not enough stock. Available:х» |
| R 5.1.3.5 | После неудачной попытки добавить продукт в корзину, в поле ввода количества должны остаться данные которые ввел пользователь. |
| R 5.1.3.6 | Поиск, сортировка и фильтрация сохраняются после добавления в корзину или неудачной попытки добавления. |
| R 5.2 | На PDP гость видит:   * Информация о продукте * Поле для ввода количества * Кнопка для добавления в корзину * Отзывы о продукте |
| R 5.2.1 | Информация о продукте включает:   * Название * Категорию * Количество на складе * Историю цены * Цену * Фото |
| R 5.2.2 | Процес добавления в корзину эдентичен пунктам R 5.1.3.2 - R 5.1.3.4 |
| R 5.2.3 | У каждого отзыва отображается:   * Автор * Оценка от 1 до 5 * Комментарий |
| R 5.2.4 | Если у продукта нет отзывов, то отображается сообщение «There are no reviews!». |
| R 5.3 | На странице корзины есть следующие элементы:   * Таблица продуктов * Кнопки “delete” * Общее количество продуктов * Сумма заказа * Кнопка “Update” * Checkout button |
| R 5.3.1 | На странице корзины пользователь может редактировать количества каждого продукта через поле ввода количества и кнопку update. Валидация эквивалентна пунктам R 5.1.3.3 - R 5.1.3.5. |
| R 5.3.2 | В случае успешного обновления корзины всплывает сообщение «Cart updated successfully» |
| R 5.3.3 | При нажатии на кнопку «Checkout» происходит перенаправление на страницу аутентификации. |
| R 5.3.4 | При нажатии на кнопку «Delete» напротив продукта, происходит удаление этого продукта из корзины и выводится сообщение «Cart item removed successfully» |

Данную часть системы может видеть пользователь аутентифицированный как покупатель. Покупатель может совершать заказы и оставлять комментарии о продуктах.

Таблица – 1. 6. Аутентифицированный пользователь (как покупатель)

|  |  |
| --- | --- |
| ID | Описание требования |
| R 6.1 | На странице корзины становится доступна кнопка “checkout” после нажатия на которую пользователь попадает на “checkout page”. |
| R 6.2 | На странице чекаута демонстрируется корзина(описано выше) без возможности редактирования. |
| R 6.3 | Кроме того, на странице чекаута есть форма для ввода данных необходимых для совершения заказа. Названия полей приведены на скриншоте. Никакое поле не может быть пустым во время нажатия кнопки “place order”. В таком случае под каждым полем выводится сообщение об ошибке. |
| R 6.4 | Поле First Name, Last Name, delivery address принимает любую строку. |
| R 6.5 | Поле «телефон» принимает только цифры, может начинается с плюса, от 7 до 11 цифр. |
| R 6.6 | Поле «дата доставки» принимает дату по шаблону yyyy-MM-dd |
| R 6.7 | Поле “Payment method” является select box, и предоставляет способы оплаты, карта либо наличные. |
| R 6.8 | Если пользователь ввел часть полей удачно, а часть неудачно, то удачные значения сохраняются в полях. |
| R 6.9 | После удачного ввода данных в форму и нажатия кнопки place order, пользователь попадает на страницу обзора совершенного заказа. На данной странице отображается   * Состав совершенного заказа (список продуктов) * Сумма заказа * Количество продуктов * Данные о пользователе введенные на странице checkout.   Данная страница не имеет полей для ввода данных |
| R 6.10 | На странице подробной информации о продукте покупатель может оставлять комментарий и оценку о продукте. При вводе пустой строки в поле комментария выпадет сообщение об ошибке “can’t be empty”. При вводе более 200 символов, выпадает сообщение “max length 200 characters”. Оценка выставляется по radio button(default 5). Далее комментарий отправляется на проверку. |

Данную часть системы может видеть пользователь с ролью админ. Админ может просматривать меню админа, редактировать, создавать, удалять продукты. А также подтверждать или отклонять комментарии пользователей.

Таблица 1.7 - Аутентифицированный пользователь (как админ)

|  |  |
| --- | --- |
| ID | Описание требования |
| R 7.1 | На главной странице администратора (adminMenu.jsp) расположено понятное пользовательское меню в виде ссылок   * Добавить новый товар * Редактировать товары * Модерировать комментарии |
| R 7.1.1 | На странице добавления товара отображаются следующие элементы:   * Текстовое поле для ввода url картинки [1-255] * Текстовое поле для ввода названия категории [1..30] * Текстовое поле для ввода описания товара [1..30] * Поле для ввода стоимости товара - число с плавающей точкой >= 0 * Поле для ввода количества товара – целое число >=0   Примечание!!! Все поля являются обязательными для заполнения |
| R 7.1.1.1 | Если администратор не заполнил все поля на странице добавления товара и нажал на кнопку “Create”, то под каждым незаполненным полем появляется сообщение “Value is required” |
| R 7.1.1.2 | Если пользователь ввёл некорректное значение стоимости (нечисловое значение, либо отрицательное), новый продукт не сохранятся в базе данных, появляется сообщение “Must be non negative”. |
| R 7.1.1.3 | Если пользователь ввёл некорректное значение количества товара (нечисловое значение, либо отрицательное), новый продукт не сохранятся в базе данных, появляется сообщение “Must be non negative int”. |
| R 7.1.1.4 | Если товар с введённым наименованием уже существует в системе, то выводится сообщение «Товар с таким наименованием уже есть. Введите другое имя товара» |
| R 7.1.1.5 | После корректного заполнения полей и нажатия на кнопку «Create», появляется сообщение «Product created successfully» |
| R 7.1.2 | На странице редактирования продуктов есть следующие элементы:   * Панель фильтрации * Таблица продуктов |
| R 7.1.2.1 | Панель фильтрации содержит поля:   * Product tag - тип string[0..30] * Min price – число с плавающей точкой >= 0 * Max price – число с плавающей точкой >= 0 * Min stock – целое число >= 0   А так же есть кнопка “Filter” |
| R 7.1.2.2 | Поля не являются обязательными для заполнения. При нажатии кнопки “Filter” с пустыми полями, будут показаны все продукты. |
| R 7.1.2.3 | При вводе не неотрицательного числа в поле “min price” or “max price” и нажатия кнопки «FILTER» выведется сообщение Must be non negative |
| R 7.1.2.4 | При вводе не неотрицательного или не целого числа в поле “min stock” и нажатия кнопки «FILTER» выведется сообщение Must be non negative int |
| R 7.1.2.5 | Таблица отображающая продукты и названия ее столбцов отображены на скриншоте. В каждой ячейке таблицы находится поле в котором хранится значение этого поля для конкретного продукта. Админ может изменять эти поля и после нажатия кнопки “Update” (расположенной под таблицей) данные о продуктах обновляются. |
| R 7.1.2.6 | На против каждого продукта есть кнопка удаления, при нажатии на которую продукт удаляется из базы данных. |
| R 7.1.2.7 | Валидация осуществляется аналогично пунктам R 7.1.1.1 - R 7.1.1.4. Сообщения об ошибках выводятся в каждой ячейке, содержащей не валидные данные. |
| R 7.1.2.8 | На странице модерации отзывов в случае их отсутствия выводится сообщение There are no reviews |
| R 7.1.2.9 | Страница модерации отзывов содержит следующую таблицу: |
| R 7.1.2.10 | Админ может нажать кнопку “approve” и тогда отзыв будет виден на PDP этого продукта, либо нажать кнопку “reject” и тогда отзыв будет удален. |
| R 7.1.2.11 | После нажатия кнопки “approve” выводится сообщение «Review approved successfully» |
| R 7.1.2.12 | После нажатия кнопки “ reject ” выводится сообщение «Review rejected successfully» |

# 2. Проектирование программных модулей

Был использован паттерн проектирования MVC. Предполагает разделение данных приложении и управляющей логики на 3 отдельных компонента. Модификация каждого осуществляется независимо. Применение шаблона на практике приводит к тому, что базовая трёхслойная архитектура распадается на многоуровневую. Причем один уровень знает о существовании только следующего за ним. Модель представляет данные и методы работы с ними.

Кроме того, были реализованы следующие паттерны: command, proxy, dto, flyweight, strategy, singleton, chain of responsibility, state.

Были соблюдены принципы SOLID, Yagni, Kiss, Dry. Пример Liskov substitution principle : подклассы не могут замещать поведение базовых классов, а могут лишь дополнять. Если s является подтипом t, тогда объекты типа t в программе могут быть заменены на объекты типа s. Если в функцию приходит родитель, то если передать наследника, все сработает. Подкласс не должен требовать от вызывающего кода больше, чем базовый класс и не должен предоставлять вызывающему коду меньше, чем базовый класс. Пример несоблюдения: parant.m() – ok; child.init(); cild.m(); - ok; But simply child.m() – error. Клиентский код не должен знать с каким типом он работает(проверка instanceof – значит принцип не соблюдается).

Иногда невозможно соблюсти LSP при наследовании, квадрат не может быть наследником прямоугольника, в противном случае LSP не соблюден.

Шаблонный метод увеличивает шансы того, что LSP будет соблюден. Программист, который наследует свой класс от класса с шаблонным методом скорее всего будет вынужден реализовать предполагаемую логику. Запустить двигатель (подключить стартер, подготовить топливо...)

Dependency inversion. Используй интерфейсы вместо конкретных реализаций. Модули зависят от абстракций.

# 3. Реализация программных модулей

Софт реализован на Java в среде разработки Intelij Idea.

Все части программы реализованы в различных классах. Были задействованы такие технологии как Tomcat, Servelt API, JSP, JSTL, Junit, Mockito, HTML, CSS, JS, и т.д.

Результатом разделения проекта по модулям стала следующая структура пакетов, классов и файлов:

* Пакет com.es.phoneshop.controller представлен на рисунке 3.1.

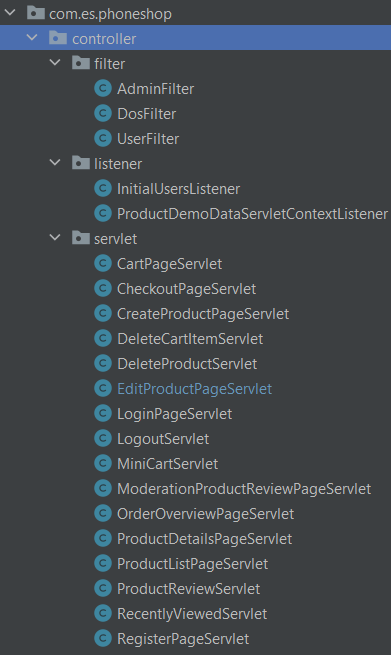
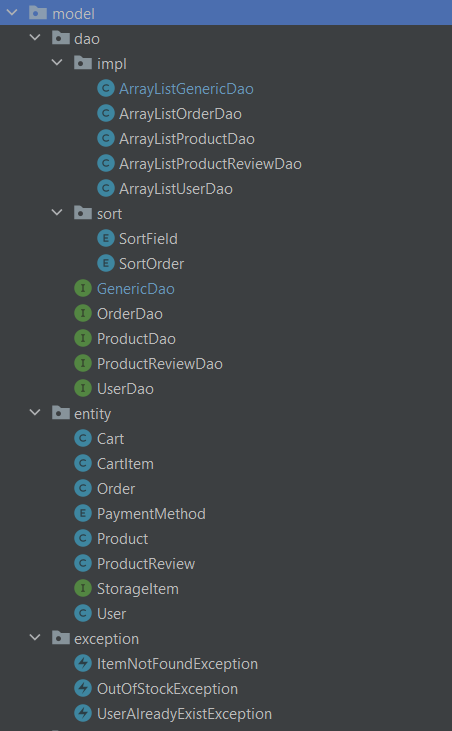


Рисунок 3.1 – Структура контроллера

* Пакет com.es.phoneshop.model представлен на рисунке 3.2.



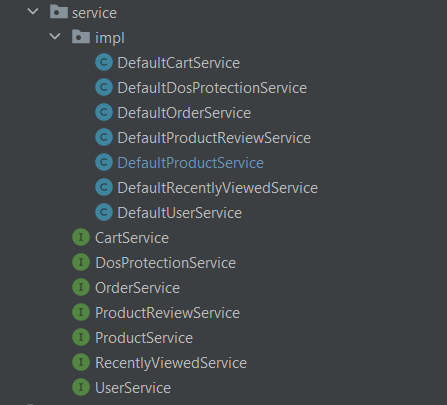


Рисунок 3.2 – Структура модели

* Пакеты com.es.phoneshop.util и com.es.phoneshop.value представлены на рисунке 3.3.



Рисунок 3.3 – Структура инструментальных пакетов

* Директория webapp представлена на рисунке 3.4.

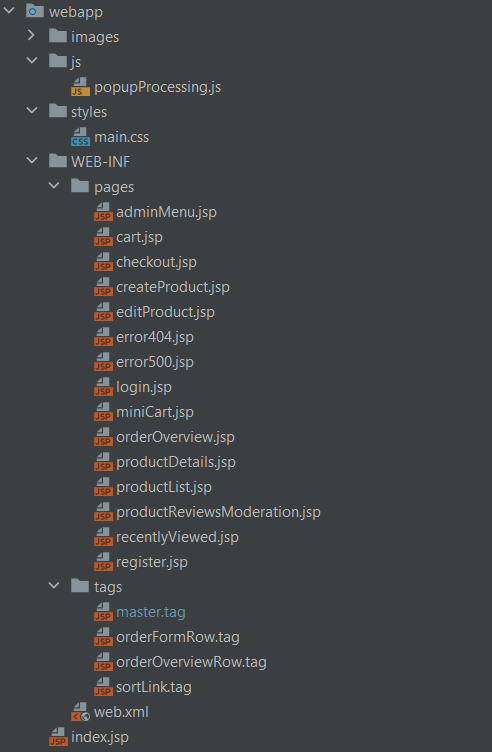


Рисунок 3.4 – Структура webapp

Контроллекр регистрации пользователей реализован как показано на рисунке 3.5.

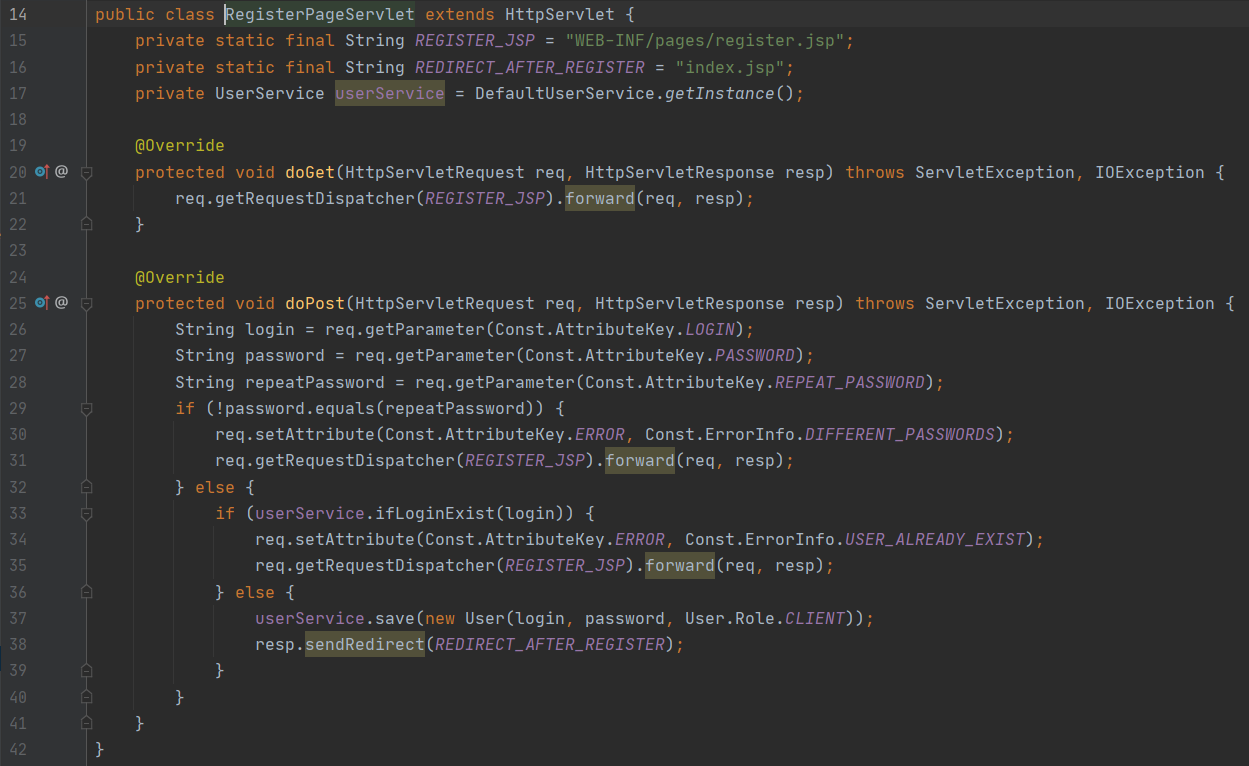


Рисунок 3.5 – Реализация RegisterPageServlet

Контроллер редактирования продуктов реализован как показано на рисунке 3.6.

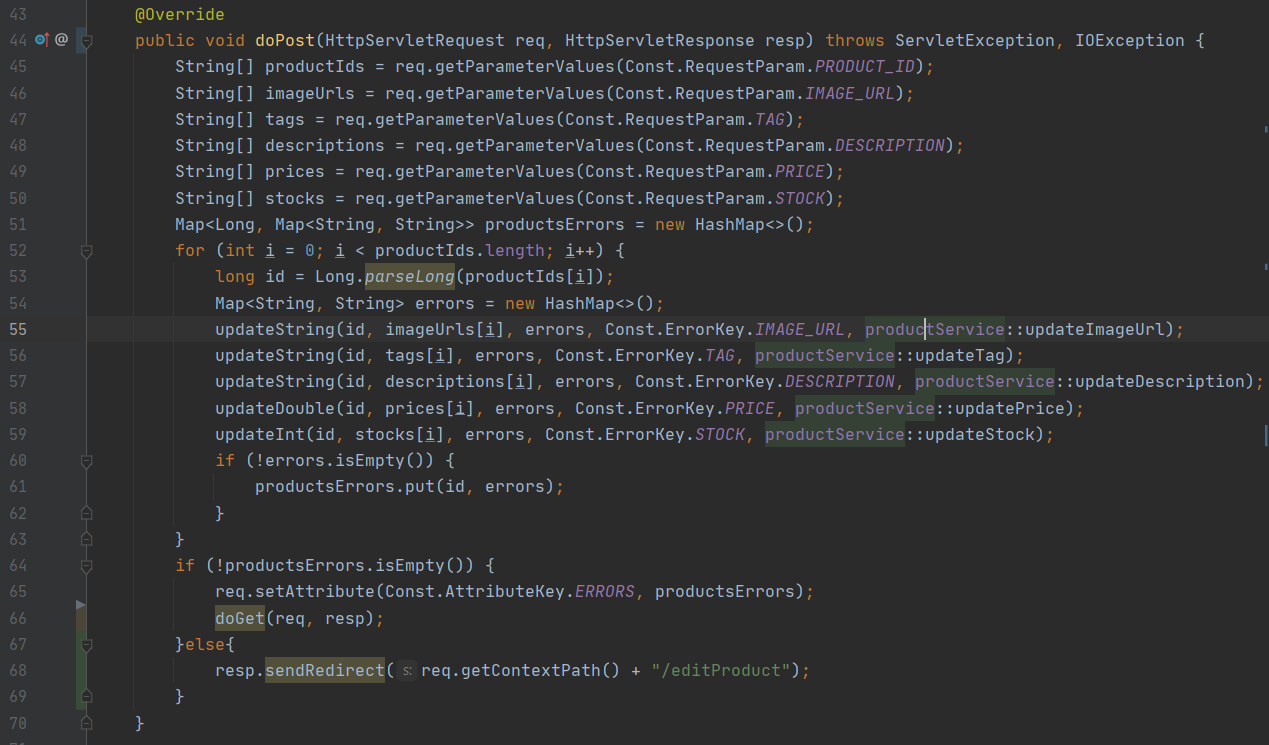


Рисунок 3.6 – Реализация EditProductPageServlet.doPost

Метод OrderOverviewPageServlet.doPost реализован как показано на рисунке 3.7.

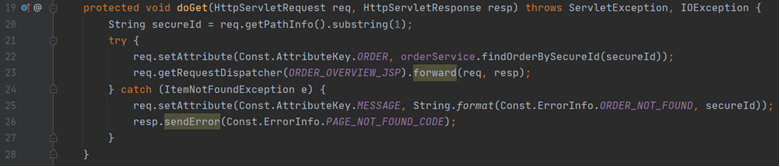


Рисунок 3.7 – Реализация DosFilter.doFilter

Фильтр защищающий от DOS атак реализован как показано на рисунке 3.8.

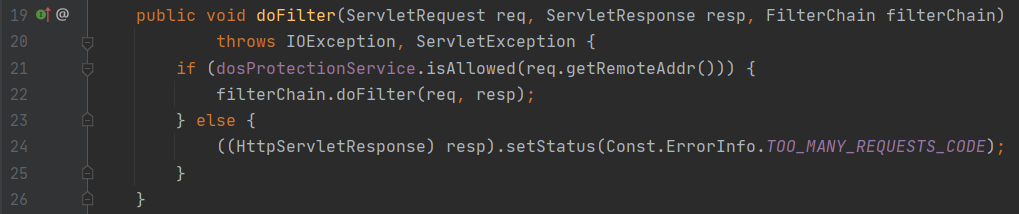


Рисунок 3.8 – Реализация DosFilter.doFilter

Метод equals для класса CardInfo реализован как показано на риснке 3.9.

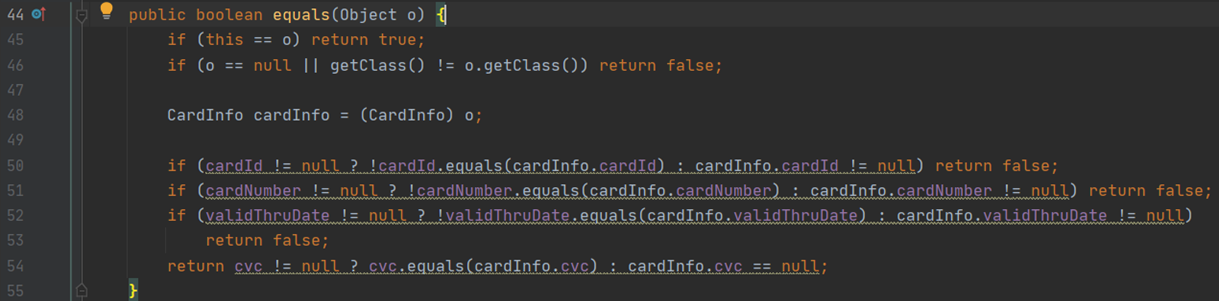


Рисунок 3.9 – Реализация CardInfo.equals

Метод фильтрации продуктов по свойствам реализован как показано на рисунке 3.10.

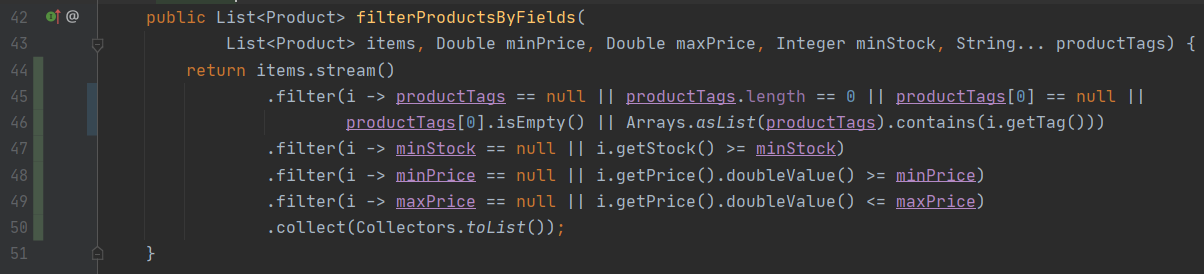


Рисунок 3.10 – Реализация DefaultProductService.filterProductsByFields

# 4. Проектирование модульных тестов

## ****4.1**** Построение ориентированных графов МакКейба

Тестовые методы сопровождются графами Маккейба. Для метода на рисунке 3.5 граф Маккейба будет выглядеть следющим образом рисунок 4.1.

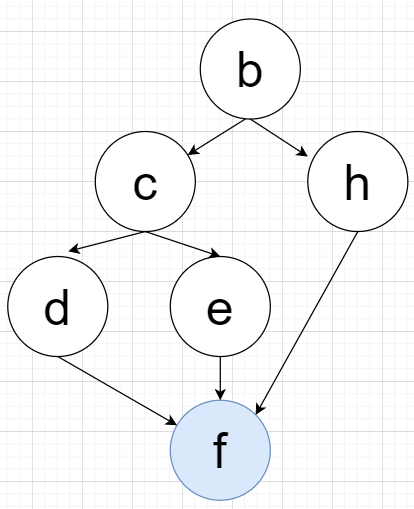


Рисунок 4.1 – Граф Маккейба для метода на рисунке 3.5

Для метода на рисунке 3.6 граф Маккейба будет выглядеть следющим образом рисунок 4.2.

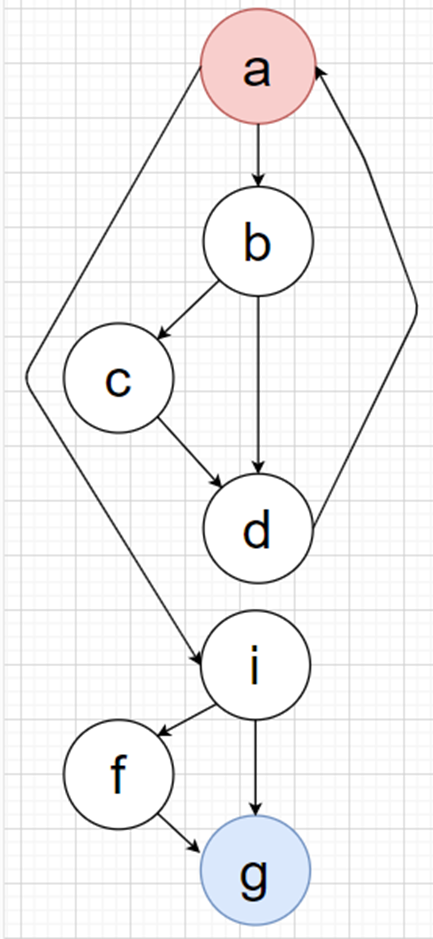


Рисунок 4.2 – Граф Маккейба для метода на рисунке 3.6

Для метода на рисунке 3.7 граф Маккейба будет выглядеть следющим образом рисунок 4.3.

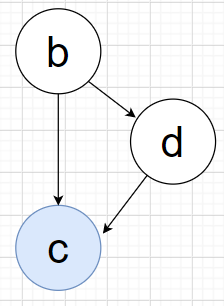


Рисунок 4.3 – Граф Маккейба для метода на рисунке 3.7

Для метода на рисунке 3.8 граф Маккейба будет выглядеть следющим образом рисунок 4.4.

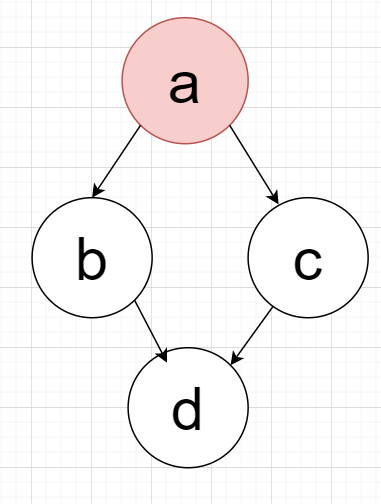


Рисунок 4.4 – Граф Маккейба для метода на рисунке 3.8

Для метода на рисунке 3.9 граф Маккейба будет выглядеть следющим образом рисунок 4.5.

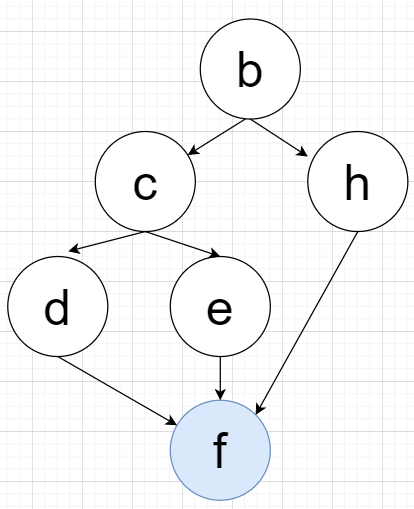


Рисунок 4.5 – Граф Маккейба для метода на рисунке 3.9

## ****4.2**** Формирование проходов

Метрический показатель сложности или цикломатическое число ***G*** потокового графа определяется по формуле **G=R-V+2*,*** где ***R*** – количество ребер графа. А ***V*** – количество вершин графа.

Для графа на рисунке 4.1 цикломатическое число будет выщитано следющим образом. G = 7 – 6 + 2 = 3. Проходы по графу приведены в таблице 4.1.

Таблица 4.1 – Проходы по графу Маккейба для рисунка 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **G** | **№ прохода** | **Описание прохода** | **Контрольные примеры, позволяющие реализовать описанную ситуацию** | **Тест пройден**  **Да/Нет** |
| **G=3** | **1**  **2**  **3** | **b-h-f**  **b-c-d-f**  **b-c-e-f** | !password.equals(repeatPassword)  password.equals(repeatPassword) && userService.ifLoginExist(login)  password.equals(repeatPassword) && !userService.ifLoginExist(login) | **да** |

Для графа на рисунке 4.2 цикломатическое число будет выщитано следющим образом. G = 9 – 7 + 2 = 4. Проходы по графу приведены в таблице 4.2.

Таблица 4.2 – Проходы по графу Маккейба для рисунка 4.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **G** | **№ прохода** | **Описание прохода** | **Контрольные примеры, позволяющие реализовать описанную ситуацию** | **Тест пройден**  **Да/Нет** |
| **G=4** | **1**  **2**  **3**  **4** | **A-b-d-a-I-g**  **A-b-d-a-I-f-g**  **A-b-c-d-a-i-g**  **A-b-c-d-a-i-f-g** | errors.isEmpty() && productsErrors.isEmpty()  errors.isEmpty() && !productsErrors.isEmpty()  !errors.isEmpty() && productsErrors.isEmpty()  !errors.isEmpty() && !productsErrors.isEmpty() | **да** |

Для графа на рисунке 4.3 цикломатическое число будет выщитано следющим образом. G= 3 – 3 + 2 = 2. Проходы по графу приведены в таблице 4.3.

Таблица 4.3 – Проходы по графу Маккейба для рисунка 4.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **G** | **№ прохода** | **Описание прохода** | **Контрольные примеры, позволяющие реализовать описанную ситуацию** | **Тест пройден**  **Да/Нет** |
| **G=2** | **1**  **2** | **B-c**  **B-d-c** | No exceptions  ItemNotFoundException has been thrown | **да** |

Для графа на рисунке 4.4 цикломатическое число будет выщитано следющим образом G = 4 – 4 + 2 = 2. Проходы по графу приведены в таблице 4.4.

Таблица 4.4 – Проходы по графу Маккейба для рисунка 4.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **G** | **№ прохода** | **Описание прохода** | **Контрольные примеры, позволяющие реализовать описанную ситуацию** | **Тест пройден**  **Да/Нет** |
| **G=2** | **1**  **2** | **A-b-d**  **A-c-d** | dosProtectionService.isAllowed(req.getRemoteAddr()) = true  dosProtectionService.isAllowed(req.getRemoteAddr()) = false | **да** |

Для графа на рисунке 4.5 цикломатическое число будет выщитано следющим образом. G = 9 – 6 + 2 = 5. Проходы по графу приведены в таблице 4.5.

Таблица 4.5 – Проходы по графу Маккейба для рисунка 4.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **G** | **№ прохода** | **Описание прохода** | **Контрольные примеры, позволяющие реализовать описанную ситуацию** | **Тест пройден**  **Да/Нет** |
| **G=5** | **1**  **2**  **3**  **4**  **5** | **A-b**  **A-c-b**  **A-c-d-b**  **A-c-d-i-b**  **A-c-d-i-f-b** | this == o  previous && o == null || getClass() != o.getClass()  previous && cardId != null ? !cardId.equals(cardInfo.cardId) : cardInfo.cardId != null  previous && cardNumber != null ? !cardNumber.equals(cardInfo.cardNumber) : cardInfo.cardNumber != null  previous && validThruDate != null ? !validThruDate.equals(cardInfo.validThruDate) : cardInfo.validThruDate != null | **да** |

# 5. Проектирование программных модулей

## 5.1 Реализация автоматизированных скриптов

Для метода на рисунке 3.5 тестовый класс представлен в Приложении Б с названием «com.es.phoneshop.controller.servlet.RegisterPageServletTest»

Для метода на рисунке 3.6 тестовый класс представлен в Приложении Б с названием «com.es.phoneshop.controller.servlet.EditProductPageServletTest»

Для метода на рисунке 3.7 тестовый класс представлен на рисунке 5.1.

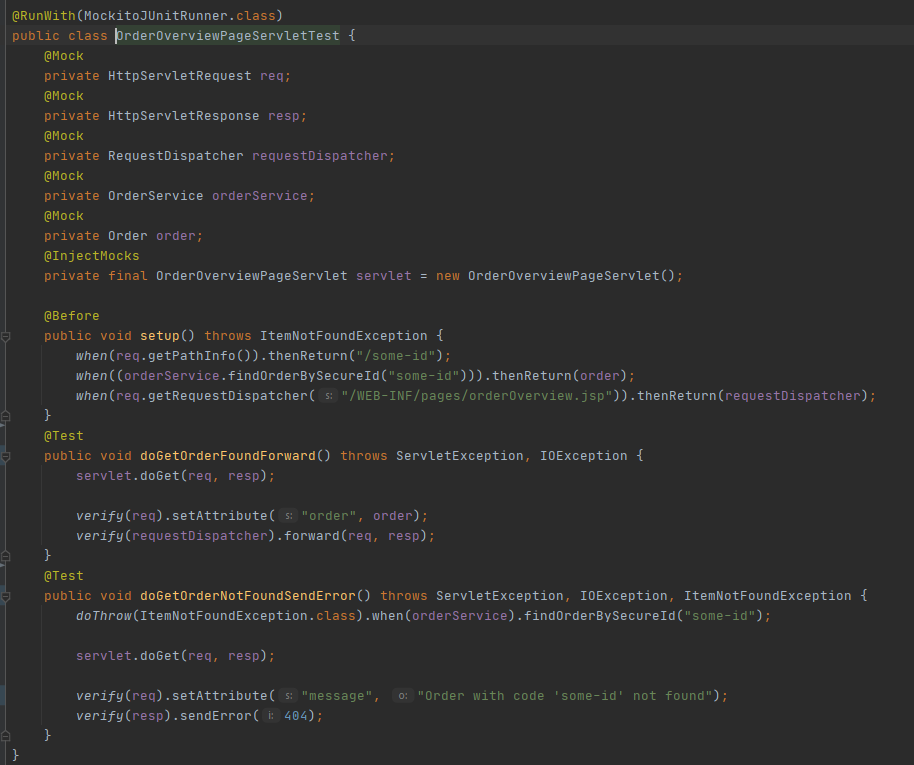


Рисунок 5.1 – Реализация тестов для кода на рисунке 3.7

Для метода на рисунке 3.8 тестовый класс представлен на рисунке 5.2.

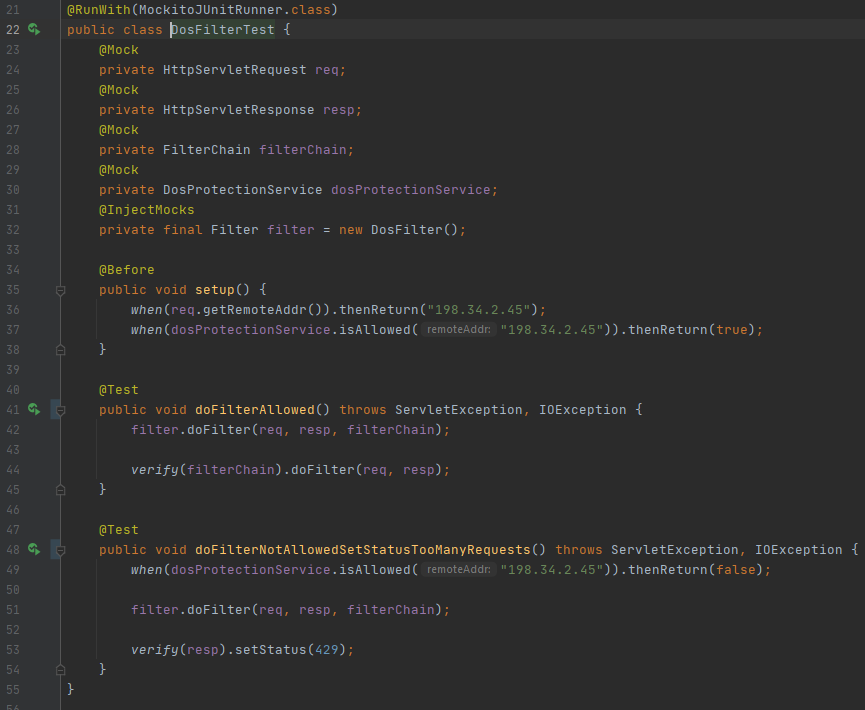


Рисунок 5.2 – Реализация тестов для кода на рисунке 3.8

Для метода на рисунке 3.9 тестовый класс представлен на рисунке 5.3**.**

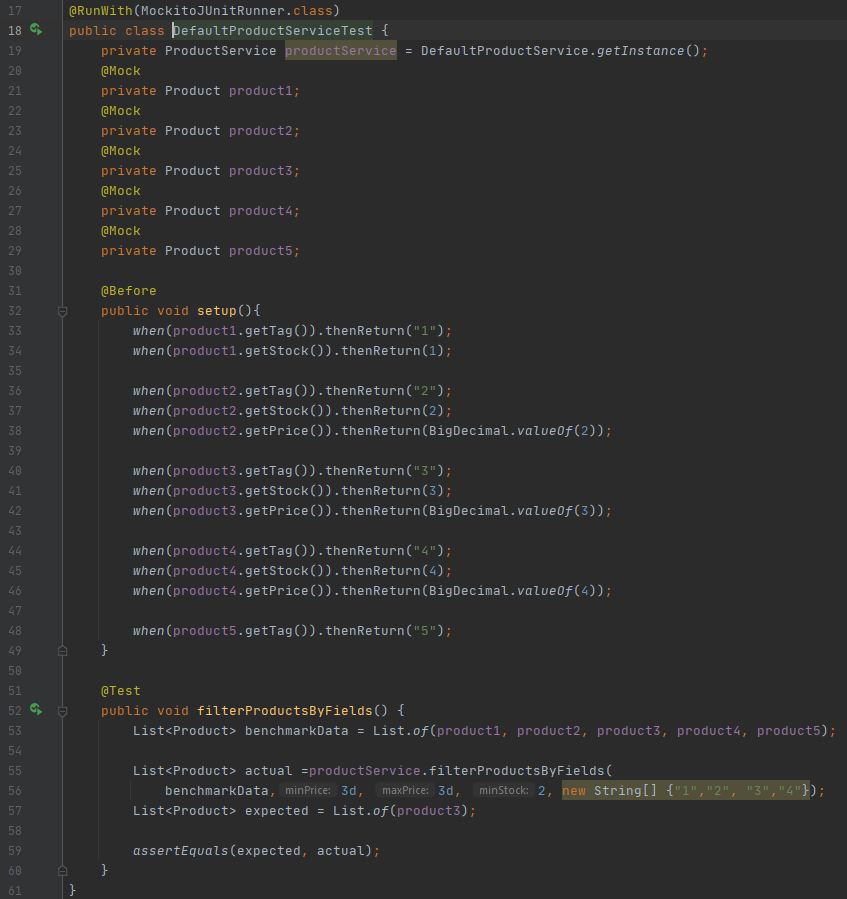


Рисунок 5.3 – Реализация тестов для кода на рисунке 3.9

### ****Интеграционное тестирование****

Тестируется взаимодействие слоев. Слои приведены на рисунке 5.4. Реализации соответствующих классов приведены в Приложении А.

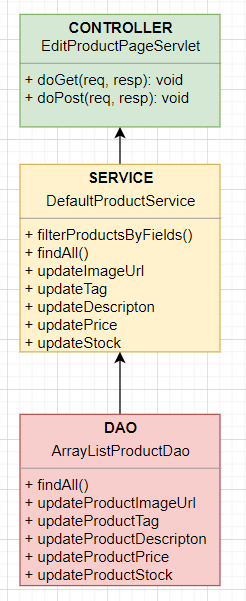


Рисунок 5.4 – Слои системы

Сервис и дао являются синглтонами.

Методы контролера:

* doGet(HttpServletRequest, HttpServletResponce) сетает в реквест список продуктов.
* doPost(HttpServletRequest, HttpServletResponce) обновляет поля продуктов (цена, количество...) удовлетворяющих параметрам запроса.

Request & respnce необходимо замокать, все остальные объеты реальные. Dao хранит список продуктов в листе. Так же дао содержит сетер для листа. В классе теста я объявлю свой лист и проинициализирую его, потом засетаю в дао. Это позволит отслеживать изменения на моем тестовом объетке листа. При тестировании doGet() проверяю какой список продуктов вернулся из сервиса. Сначала теслирую без параметров поиска, потом с параметрами. Когда тестирую doPost() сравниваю лист, который я засетал в дао (после вызова метода он должен изменится) с предполагаемым листом. Так же тестирую с неверным параметром.

Реализация тестов приведена в приложении Б под именем «**com.es.phoneshop.integration.EditProductIntegrationTest**»

## 5.2 Анализ результатов тестов

В результате проведения проверки качества были разработаны тесты, которые покрывают 100% строк тестируемых методов. В ходе тестирования была обнаружена и исправлена одна ошибка в методе EditProductPageServlet.doPost . После успешного выполнения doPost(user entered valid data) должен следовать редирект а не форвард(doGet). Дальнейшее тестирование проводилось по исправленному методу.

Было протестировано взаимодействие трех модулей программы. Модули были предварительно протестированы отдельно. Программа работает как предполагается.

## 5.3 Отладка тестов

Результаты запуска всех тестов сразу приведены на рисунках 5.5, 5.6, 5.7.

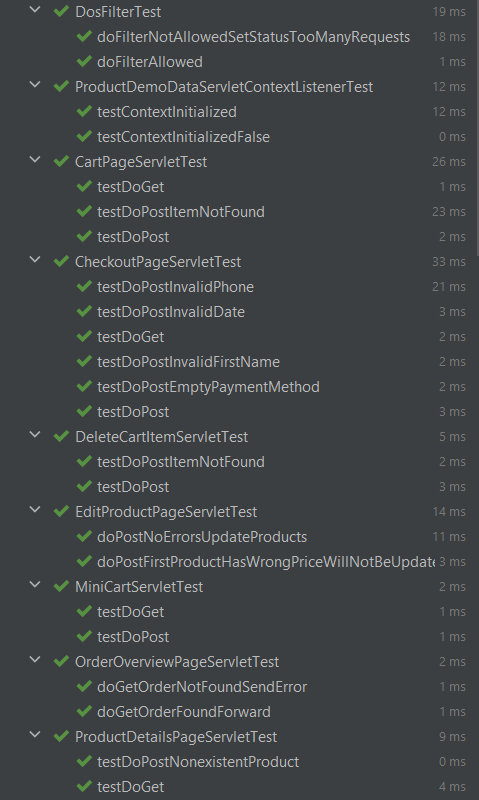


Рисунок 5.5 – Выполнение тестов часть 1

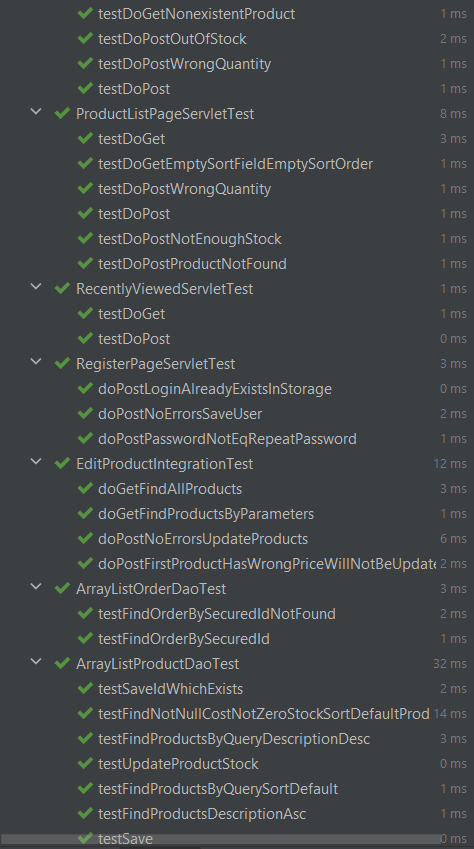


Рисунок 5.6 – Выполнение тестов часть 2



Рисунок 5.7 – Выполнение тестов часть 3

# ЗАКЛЮЧЕНИЕ

Разработанный софт позволяет повысить эффективность продаж мобильных телефонов через интернет. Программа автоматизирует работу продавца, тем самым позволяет бизнесу сократить издержи.

Реализован весь функционал в соответствии с техническим заданием к проекту. Высокий уровень покрытия тестами свидетельствует о качественном программном обеспечении. Была использована методология разработки TDD. Модульное тестирование проводилось с использованием заглушек. Кроме того, были проведены интеграционные тесты.

Приложение демонстрирует высокий уровень безопасности. Приложение реализует защиту от DOS атак. Данные пользователей надежно хранятся в хранилище.

Приятный дизайн front end части позволит привлечь еще больше клиентов, тем самым увеличивает продажи. Для реализации pop up элементов и валидации на клиенте использован JavaScript.

Успешные тесты и стабильная работа, продемонстрированная на испытаниях, говорит о готовности программы к релизу и запуску ее первой пробной версии.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Java. Сетевое программирование / В. Кумар [и др.]. – М.: Лори, 2007.- 400 с.
2. Фурунжиев Р.И. – Устройство Tomcat. Монография. - Мн.: Выш.шк, 2013. - 452 с.
3. Орлов, С.А. Организация ЭВМ и систем: Учебник для вузов / С.А. Орлов, Б.Я. Цилькер. - 2-е изд. - СПб: Питер, 2011. - 688с.
4. Фурунжиев Р.И., Гурский Н.Н. - Применение математических методов и ЭВМ. Программное моделирование систем. - Мн.: Выш. шк. 1991. – 250 с.
5. C.Бобровский - Java8. Учебный курс.

# Приложение А. Листинг исходных кодов

**com.es.phoneshop.controller.servlet.EditProductPageServlet**

public class EditProductPageServlet extends HttpServlet {  
 private static final String *ADVANCED\_SEARCH\_JSP* = "WEB-INF/pages/editProduct.jsp";  
 private ProductService productService = DefaultProductService.*getInstance*();  
  
 @Override  
 public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 String productCode = req.getParameter(Const.RequestParam.*SEARCH\_TAG*);  
 String potentialMinPrice = req.getParameter(Const.RequestParam.*MIN\_PRICE*);  
 String potentialMaxPrice = req.getParameter(Const.RequestParam.*MAX\_PRICE*);  
 String potentialMinStock = req.getParameter(Const.RequestParam.*SEARCH\_STOCK*);  
 Map<String, String> errors = new HashMap<>();  
 Double minPrice = handleNonNegativeDouble(potentialMinPrice, Const.ErrorKey.*MIN\_PRICE*, errors);  
 Double maxPrice = handleNonNegativeDouble(potentialMaxPrice, Const.ErrorKey.*MAX\_PRICE*, errors);  
 Integer minStock = handleNonNegativeInt(potentialMinStock, Const.ErrorKey.*STOCK*, errors);  
 if (errors.isEmpty()) {  
 req.setAttribute(Const.AttributeKey.*PRODUCTS*, productService.filterProductsByFields(  
 productService.findAll(), minPrice, maxPrice, minStock, productCode));  
 } else {  
 req.setAttribute(Const.AttributeKey.*SEARCH\_ERRORS*, errors);  
 }  
 req.getRequestDispatcher(*ADVANCED\_SEARCH\_JSP*).forward(req, resp);  
 }  
  
 @Override  
 public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 String[] productIds = req.getParameterValues(Const.RequestParam.*PRODUCT\_ID*);  
 String[] imageUrls = req.getParameterValues(Const.RequestParam.*IMAGE\_URL*);  
 String[] tags = req.getParameterValues(Const.RequestParam.*TAG*);  
 String[] descriptions = req.getParameterValues(Const.RequestParam.*DESCRIPTION*);  
 String[] prices = req.getParameterValues(Const.RequestParam.*PRICE*);  
 String[] stocks = req.getParameterValues(Const.RequestParam.*STOCK*);  
 Map<Long, Map<String, String>> productsErrors = new HashMap<>();  
 for (int i = 0; i < productIds.length; i++) {  
 long id = Long.*parseLong*(productIds[i]);  
 Map<String, String> errors = new HashMap<>();  
 updateString(id, imageUrls[i], errors, Const.ErrorKey.*IMAGE\_URL*, productService::updateImageUrl);  
 updateString(id, tags[i], errors, Const.ErrorKey.*TAG*, productService::updateTag);  
 updateString(id, descriptions[i], errors, Const.ErrorKey.*DESCRIPTION*, productService::updateDescription);  
 updateDouble(id, prices[i], errors, Const.ErrorKey.*PRICE*, productService::updatePrice);  
 updateInt(id, stocks[i], errors, Const.ErrorKey.*STOCK*, productService::updateStock);  
 if (!errors.isEmpty()) {  
 productsErrors.put(id, errors);  
 }  
 }  
 if (!productsErrors.isEmpty()) {  
 req.setAttribute(Const.AttributeKey.*ERRORS*, productsErrors);  
 doGet(req, resp);  
 }else{  
 resp.sendRedirect(req.getContextPath() + "/editProduct");  
 }  
 }  
  
 public Double handleNonNegativeDouble(String potentialDouble, String errorKey, Map<String, String> errors) {  
 if (potentialDouble == null || potentialDouble.isEmpty()) {  
 return null;  
 }  
 Optional<Double> number = CustomParser.*parseNonNegativeDouble*(potentialDouble);  
 if (number.isEmpty()) {  
 errors.put(errorKey, Const.ErrorInfo.*NON\_NEGATIVE\_NUMBER*);  
 }  
 return number.orElse(null);  
 }  
  
 public Integer handleNonNegativeInt(String potentialInteger, String errorKey, Map<String, String> errors) {  
 if (potentialInteger == null || potentialInteger.isEmpty()) {  
 return null;  
 }  
 Optional<Integer> number = CustomParser.*parseNonNegativeInt*(potentialInteger);  
 if (number.isEmpty()) {  
 errors.put(errorKey, Const.ErrorInfo.*NON\_NEGATIVE\_INT*);  
 }  
 return number.orElse(null);  
 }  
  
 private void updateString(long productId, String imageUrl, Map<String, String> error, String errorKey,  
 BiConsumer<Long, String> consumer) {  
 if (imageUrl.isEmpty()) {  
 error.put(errorKey, Const.ErrorInfo.*VALUE\_IS\_REQUIRED*);  
 } else {  
 consumer.accept(productId, imageUrl);  
 }  
 }  
  
 private void updateDouble(long productId, String potentialDouble, Map<String, String> error, String errorKey,  
 ObjDoubleConsumer<Long> consumer) {  
 CustomParser.*parseNonNegativeDouble*(potentialDouble)  
 .ifPresentOrElse(d -> consumer.accept(productId, d),  
 () -> error.put(errorKey, Const.ErrorInfo.*NON\_NEGATIVE\_NUMBER*));  
 }  
  
 private void updateInt(long productId, String potentialInt, Map<String, String> error, String errorKey,  
 ObjIntConsumer<Long> consumer) {  
 CustomParser.*parseNonNegativeInt*(potentialInt)  
 .ifPresentOrElse(i -> consumer.accept(productId, i),  
 () -> error.put(errorKey, Const.ErrorInfo.*NON\_NEGATIVE\_INT*));  
 }  
}

**com.es.phoneshop.model.service.impl.DefaultProductService**

public class DefaultProductService implements ProductService {  
 private ArrayListProductDao productDao = ArrayListProductDao.*getInstance*();  
  
 private DefaultProductService() {  
 }  
  
 private static class DefaultAdvancedSearchServiceHolder {  
 private static final DefaultProductService *DEFAULT\_PRODUCT\_SERVICE\_INSTANCE* =  
 new DefaultProductService();  
 }  
  
 public static DefaultProductService getInstance() {  
 return DefaultAdvancedSearchServiceHolder.*DEFAULT\_PRODUCT\_SERVICE\_INSTANCE*;  
 }  
  
 @Override  
 public void save(Product product) {  
 productDao.save(product);  
 }  
  
 @Override  
 public List<Product> findProducts(String query, SortField sortField, SortOrder sortOrder) {  
 return productDao.findProducts(query, sortField, sortOrder);  
 }  
  
 @Override  
 public List<Product> filterProductsByFields(  
 List<Product> items, Double minPrice, Double maxPrice, Integer minStock, String... productTags) {  
 return items.stream()  
 .filter(i -> productTags == null || productTags.length == 0 || productTags[0] == null ||  
 productTags[0].isEmpty() || Arrays.*asList*(productTags).contains(i.getTag()))  
 .filter(i -> minStock == null || i.getStock() >= minStock)  
 .filter(i -> minPrice == null || i.getPrice().doubleValue() >= minPrice)  
 .filter(i -> maxPrice == null || i.getPrice().doubleValue() <= maxPrice)  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public List<String> findTags() {  
 return productDao.findTags();  
 }  
  
 @Override  
 public Product find(long productId) throws ItemNotFoundException {  
 return productDao.find(productId);  
 }  
  
 @Override  
 public List<Product> findAll() {  
 return productDao.findAll();  
 }  
  
 @Override  
 public void saveProduct(String imageUrl, String productCode, String description, BigDecimal price, int stock) {  
 Product product = new Product();  
 product.setImageUrl(imageUrl);  
 product.setTag(productCode);  
 product.setDescription(description);  
 product.setPrice(price);  
 product.setStock(stock);  
 product.setCurrency(Currency.*getInstance*("USD"));  
 productDao.save(product);  
 }  
  
 @Override  
 public void delete(long productId) {  
 productDao.delete(productId);  
 }  
  
 @Override  
 public void updateImageUrl(long productId, String imageUrl) {  
 productDao.updateProductImageUrl(productId, imageUrl);  
 }  
  
 @Override  
 public void updateTag(long productId, String tag) {  
 productDao.updateProductTag(productId, tag);  
 }  
  
 @Override  
 public void updateDescription(long productId, String description) {  
 productDao.updateProductDescription(productId, description);  
 }  
  
 @Override  
 public void updatePrice(long productId, double price) {  
 productDao.updateProductPrice(productId, price);  
 }  
  
 @Override  
 public void updateStock(long productId, int stock) {  
 productDao.updateProductStock(productId, stock);  
 }  
}

**com.es.phoneshop.model.dao.impl.ArrayListProductDao**

public class ArrayListProductDao extends ArrayListGenericDao<Product> implements ProductDao {  
 private static final String *BLANK* = "\\p{Blank}";  
  
 private ArrayListProductDao() {  
 }  
  
 private static class SingletonHolder {  
 public static final ArrayListProductDao *ARRAY\_LIST\_PRODUCT\_DAO\_INSTANCE* = new ArrayListProductDao();  
  
 private SingletonHolder() {  
 }  
 }  
  
 public static ArrayListProductDao getInstance() {  
 return SingletonHolder.*ARRAY\_LIST\_PRODUCT\_DAO\_INSTANCE*;  
 }  
  
 @Override  
 public synchronized void updateProductImageUrl(long productId, String imageUrl) {  
 items.stream().filter(p -> p.getId().equals(productId)).findAny().ifPresent(p -> p.setImageUrl(imageUrl));  
 }  
  
 @Override  
 public synchronized void updateProductDescription(long productId, String description) {  
 items.stream().filter(p -> p.getId().equals(productId)).findAny().ifPresent(p -> p.setDescription(description));  
 }  
  
 @Override  
 public synchronized void updateProductTag(long productId, String tag) {  
 items.stream().filter(p -> p.getId().equals(productId)).findAny().ifPresent(p -> p.setTag(tag));  
 }  
  
 @Override  
 public synchronized void updateProductPrice(long productId, double price) {  
 items.stream().filter(i -> i.getId().equals(productId)).findAny()  
 .ifPresent(p -> {  
 p.setPrice(BigDecimal.*valueOf*(price));  
 p.setPriceHistory(LocalDate.*now*(), BigDecimal.*valueOf*(price));  
 });  
 }  
  
 @Override  
 public synchronized void updateProductStock(long productId, int stockValue) {  
 items.stream().filter(i -> i.getId().equals(productId)).findAny().ifPresent(p -> p.setStock(stockValue));  
 }  
  
 @Override  
 public synchronized List<String> findTags() {  
 return items.stream().map(Product::getTag).distinct().collect(Collectors.*toList*());  
 }  
  
 @Override  
 public synchronized List<Product> findProducts(String query, SortField sortField, SortOrder sortOrder) {  
 if (sortField != SortField.*DEFAULT* && sortOrder != SortOrder.*DEFAULT*) {  
 Comparator<Product> comparator = Comparator.*comparing*(p -> defineSortField(p, sortField));  
 if (sortOrder == SortOrder.*DESC*) {  
 comparator = comparator.reversed();  
 }  
 return search(query).sorted(comparator).collect(Collectors.*toList*());  
 }  
 return search(query).collect(Collectors.*toList*());  
 }  
  
 private <U> U defineSortField(Product p, SortField sortField) {  
 return (U) (SortField.*DESCRIPTION* == sortField ? p.getDescription() : p.getPrice());  
 }  
  
 private Stream<Product> search(String query) {  
 return items.stream()  
 .filter(getPriceAndStockPredicate(StringUtils.*isBlank*(query)))  
 .filter(p -> Arrays.*stream*(query.split(*BLANK*)).anyMatch(p.getDescription()::contains))  
 .sorted(Comparator.*comparing*(p -> Arrays.*stream*(query.split(*BLANK*))  
 .filter(p.getDescription()::contains).count(), Comparator.*reverseOrder*()));  
 }  
  
 private Predicate<Product> getPriceAndStockPredicate(boolean queryPresent) {  
 if (queryPresent) {  
 return product -> product.getPrice() != null && product.getStock() > 0;  
 } else {  
 return product -> true;  
 }  
 }  
}

# Приложение Б. Листинг автоматизированных скриптов

**com.es.phoneshop.controller.servlet.RegisterPageServletTest**

@RunWith(MockitoJUnitRunner.class)

public class RegisterPageServletTest {

@Mock

private HttpServletRequest request;

@Mock

private HttpServletResponse response;

@Mock

private RequestDispatcher requestDispatcher;

@Mock

private final UserService userService = DefaultUserService.getInstance();

@InjectMocks

private final RegisterPageServlet servlet = new RegisterPageServlet();

@Before

public void setup() {

when(request.getParameter("login")).thenReturn("ivan");

when(request.getParameter("password")).thenReturn("qwerty");

when(request.getParameter("repeatPassword")).thenReturn("qwerty");

when(userService.ifLoginExist("ivan")).thenReturn(false);

when(request.getRequestDispatcher("WEB-INF/pages/register.jsp")).thenReturn(requestDispatcher);

}

@Test

public void doPostNoErrorsSaveUser() throws ServletException, IOException {

servlet.doPost(request, response);

verify(userService).save(new User("ivan", "qwerty", User.Role.CLIENT));

verify(response).sendRedirect("index.jsp");

}

@Test

public void doPostPasswordNotEqRepeatPassword() throws ServletException, IOException {

when(request.getParameter("repeatPassword")).thenReturn("qwerty1");

servlet.doPost(request, response);

verify(request).setAttribute("error", "You entered different passwords.");

verify(requestDispatcher).forward(request, response);

}

@Test

public void doPostLoginAlreadyExistsInStorage() throws ServletException, IOException {

when(userService.ifLoginExist("ivan")).thenReturn(true);

servlet.doPost(request, response);

verify(request).setAttribute("error", "Such a user already exists. Try another login.");

verify(requestDispatcher).forward(request, response);

}

}

**com.es.phoneshop.controller.servlet.EditProductPageServletTest**

@RunWith(MockitoJUnitRunner.class)

public class EditProductPageServletTest {

@Mock

private HttpServletRequest request;

@Mock

private HttpServletResponse response;

@Mock

private RequestDispatcher requestDispatcher;

@Mock

private final ProductService productService = DefaultProductService.getInstance();

@InjectMocks

private final EditProductPageServlet servlet = new EditProductPageServlet();

@Before

public void setup() {

when(request.getContextPath()).thenReturn("contextPath");

when(request.getParameterValues("productId")).thenReturn(new String[]{"1", "2"});

when(request.getParameterValues("imageUrl")).thenReturn(new String[]{"1url", "2url"});

when(request.getParameterValues("tag")).thenReturn(new String[]{"htc", "palm"});

when(request.getParameterValues("description")).thenReturn(new String[]{"d1", "d2"});

when(request.getParameterValues("price")).thenReturn(new String[]{"1", "2"});

when(request.getParameterValues("stock")).thenReturn(new String[]{"1", "2"});

when(request.getRequestDispatcher("WEB-INF/pages/editProduct.jsp")).thenReturn(requestDispatcher);

}

@Test

public void doPostNoErrorsUpdateProducts() throws ServletException, IOException {

servlet.doPost(request, response);

verify(productService).updateImageUrl(1, "1url");

verify(productService).updateTag(1, "htc");

verify(productService).updateDescription(1, "d1");

verify(productService).updatePrice(1, 1);

verify(productService).updateStock(1, 1);

verify(productService).updateImageUrl(2, "2url");

verify(productService).updateTag(2, "palm");

verify(productService).updateDescription(2, "d2");

verify(productService).updatePrice(2, 2);

verify(productService).updateStock(2, 2);

verify(response).sendRedirect("contextPath/editProduct");

}

@Test

public void doPostFirstProductHasWrongPriceWillNotBeUpdated() throws ServletException, IOException {

when(request.getParameterValues("price")).thenReturn(new String[]{"notParsed", "2"});

servlet.doPost(request, response);

verify(productService).updateImageUrl(1, "1url");

verify(productService).updateTag(1, "htc");

verify(productService).updateDescription(1, "d1");

verify(productService, never()).updatePrice(eq(1), anyDouble());

verify(productService).updateStock(1, 1);

verify(productService).updateImageUrl(2, "2url");

verify(productService).updateTag(2, "palm");

verify(productService).updateDescription(2, "d2");

verify(productService).updatePrice(2, 2);

verify(productService).updateStock(2, 2);

verify(requestDispatcher).forward(request, response);

}

}

**com.es.phoneshop.integration.EditProductIntegrationTest**

@RunWith(MockitoJUnitRunner.class)

public class EditProductIntegrationTest {

@Mock

private HttpServletRequest request;

@Mock

private HttpServletResponse response;

@Mock

private RequestDispatcher requestDispatcher;

private final EditProductPageServlet servlet = new EditProductPageServlet();

private ProductDao productDao = ArrayListProductDao.getInstance();

private Currency usd = Currency.getInstance("USD");

private List<Product> products;

private List<Product> getSampleProducts() {

List<Product> productList = List.of(

new Product("Samsung", "Samsung Galaxy S", new BigDecimal(100), usd, 100, "a"),

new Product("Samsung", "Samsung Galaxy S II", new BigDecimal(200), usd, 0, "a"),

new Product("Samsung", "Samsung Galaxy S III", new BigDecimal(300), usd, 5, "a"),

new Product("Apple", "Apple iPhone", new BigDecimal(200), usd, 10, "a"));

for (int i = 0; i < productList.size(); i++) {

productList.get(i).setId((long) i + 1);

}

return productList;

}

@Before

public void setup() {

products = getSampleProducts();

productDao.setItems(products);

when(request.getContextPath()).thenReturn("contextPath");

when(request.getParameterValues("productId")).thenReturn(new String[]{"1", "2"});

when(request.getParameterValues("imageUrl")).thenReturn(new String[]{"1url", "2url"});

when(request.getParameterValues("tag")).thenReturn(new String[]{"htc", "palm"});

when(request.getParameterValues("description")).thenReturn(new String[]{"d1", "d2"});

when(request.getParameterValues("price")).thenReturn(new String[]{"1", "2"});

when(request.getParameterValues("stock")).thenReturn(new String[]{"1", "2"});

when(request.getRequestDispatcher("WEB-INF/pages/editProduct.jsp")).thenReturn(requestDispatcher);

}

@Test

public void doGetFindAllProducts() throws ServletException, IOException {

servlet.doGet(request, response);

verify(request).setAttribute("products", getSampleProducts());

}

@Test

public void doGetFindProductsByParameters() throws ServletException, IOException {

when(request.getParameter("searchTag")).thenReturn("Samsung");

when(request.getParameter("minPrice")).thenReturn("70");

when(request.getParameter("maxPrice")).thenReturn("200");

when(request.getParameter("searchStock")).thenReturn("80");

Product soughtProduct = new Product("Samsung", "Samsung Galaxy S", new BigDecimal(100),

usd, 100, "a");

soughtProduct.setId(1l);

servlet.doGet(request, response);

verify(request).setAttribute("products", List.of(soughtProduct));

}

@Test

public void doPostNoErrorsUpdateProducts() throws ServletException, IOException {

List<Product> actual = products;

List<Product> expected = List.of(

new Product("htc", "d1", BigDecimal.valueOf(1d), usd, 1, "1url"),

new Product("palm", "d2", BigDecimal.valueOf(2d), usd, 2, "2url"),

new Product("Samsung", "Samsung Galaxy S III", new BigDecimal(300), usd, 5, "a"),

new Product("Apple", "Apple iPhone", new BigDecimal(200), usd, 10, "a"));

for (int i = 0; i < expected.size(); i++) {

expected.get(i).setId((long) i + 1);

}

servlet.doPost(request, response);

assertEquals(expected, actual);

}

@Test

public void doPostFirstProductHasWrongPriceWillNotBeUpdated() throws ServletException, IOException {

when(request.getParameterValues("price")).thenReturn(new String[]{"notParsed", "2"});

List<Product> actual = products;

List<Product> expected = List.of(

new Product("htc", "d1", BigDecimal.valueOf(100), usd, 1, "1url"),

new Product("palm", "d2", BigDecimal.valueOf(2d), usd, 2, "2url"),

new Product("Samsung", "Samsung Galaxy S III", new BigDecimal(300), usd, 5, "a"),

new Product("Apple", "Apple iPhone", new BigDecimal(200), usd, 10, "a"));

for (int i = 0; i < expected.size(); i++) {

expected.get(i).setId((long) i + 1);

}

servlet.doPost(request, response);

assertEquals(expected, actual);

}

}