# РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## 4.1 Быстрое добавление в корзину

После авторизации в системе, пользователь имеет возможность совершить быстрое добавление в корзину, что особенно полезно для бизнес-покупателей.

При переходе на страницу быстрого заказа, пользователь получает возможность ввести необходимую ему информацию в форму, в которой требуется заполнить поля с названием модели телефона и количеством нужных устройств. После нажатия на кнопку, клиент отправляет запрос на сервер. Изначально происходит попытка добавления в корзину каждого элемента введённого в форме:

quickOrderElementsDto

.getQuickOrderElements()

.forEach(quickOrderElement -> {  
 int indexInList = quickOrderElementsDto

.getQuickOrderElements

.indexOf(quickOrderElement);

addToCart(quickOrderElementsDto, cart, bindingResult, indexInList);  
});

При вызове метода addToCart, происходит определение значений для определённого устройства, а также эти значения проверяются на пустоту:

final Long quantity = quickOrderElement

.getQuantity();  
final String modelPhone = quickOrderElement

.getModel();

if (quantity == null &&modelPhone

.isEmpty())  
 {return;}

После успешной проверки значений на пустоту, сервер запросит данные об устройстве в базе данных, поиск производится по названию модели устройства:

final Optional<Phone> optionalPhone = phoneServiceImpl

.getPhoneByModel(modelPhone);

После получения ответа в предыдущей функции, изначально происходит проверка на заполненность результата, если результат пуст, функция возвращает управление другой части программы. Затем происходит

проверка на заполненность поля quantity. Затем происходит попытка

добавления элемента в корзину и элемент стирается из списка.

optionalPhone

.ifPresent(phone ->  
 {  
 if (quantity != null)  
 {  
 try  
 {  
 httpSessionCartService

.addPhone(phone.getId(), quantity, cart);

quickOrderElementsDto

.getQuickOrderElements()

.set(indexInList, new QuickOrderElement());  
 }  
 catch (RuntimeException e)  
 {  
 bindingResult.rejectValue("quickOrderElements[" + indexInList + "].quantity",  
 "errors.quantity.outOfStock", "Out of stock!");  
 }  
 phoneList.add(phone);  
 }  
 else  
 {  
 bindingResult.rejectValue("quickOrderElements[" + indexInList + "].quantity",  
 "errors.emptyField", "Empty field!");  
 }  
});

В случае, если устройство не было найдено пополняется список ошибок, с информированием, что продукт не найден.

if (!optionalPhone

.isPresent())  
 {  
 bindingResult.rejectValue("quickOrderElements[" + indexInList + "].model",  
 "errors.product.notFound ", "No such product!");  
 }

После завершения попыток добавления продуктов в корзину, сформирован список успешно добавленных устройств, а также список ошибок возникших при добавлении определённых устройств в корзину. Если ошибки присутствуют, то на странице быстрого заказа, формируется информация, какие устройства были добавлены, и наоборот.

if (bindingResult.hasFieldErrors())  
{  
 model.addAttribute("successfulPhones", phoneList);  
 model.addAttribute("errors", bindingResult);  
 return PhoneshopPages.UserPages.*QuickOrderPage*;  
}

**4.2 Регистрация пользователя**

При открытии приложения, неавторизованный пользователь может просматривать информацию о товарах, искать нужный ему продукт. Однако для совершения каких-либо действий направленных на заказ устройств, ему требуется зарегистрироваться и авторизоваться.

Процесс регистрации начинается с заполнения нужных полей на странице для регистрации. После нажатия на кнопку отправляется POST-запрос на сервер и происходит его обработка. Первостепенно поля переданные от клиента проверяются на соответствие требованиям:

if (username == null || username.isEmpty() || password1 == null || password1.isEmpty() || username.length() < 5 || password1.length() < 5)  
{  
 model.addAttribute("errorEmpty", true);  
 return PhoneshopPages.UserPages.RegisterPage;  
}

В случае несовпадения какого-либо из требований, пользователю будет выведено сообщение, о несовпадении требований.

Так же требования проверяются на клиентской части приложения с помощью языка JavaScript. В первую очередь проверяется соответствие строки пароля заданным требованиям:

if (password1 == '' || password1.length < 5)  
 $('#password1-err').fadeOut(700, function () {  
 $(this).text('Password cant be empty and it should be >5 symbols').fadeIn();  
 })

В случае пустого или слишком короткого пароля, будет отображено сообщение, о неправильном формате пароля. При регистрации пользователя требуется повторить пароль, чтобы избежать ситуации с ошибочным паролем (не такой, который хотел задать пользователь). Повтор пароля проверяется аналогично:

else if (password2 == '' || password2.length < 5)  
 $('#password2-err').fadeOut(700, function () {  
 $(this).text('Password cant be empty and it should be >5 symbols').fadeIn();  
 })

После этого, если два поля соответствуют требованиям, они будут сравнены на идентичность, если пароли не совпадут, будет отображено сообщение о несовпадении паролей:

else if (password1 != password2) {  
 $('#password-mismatch').fadeOut(700, function () {  
 $(this).text('Password mismatch').fadeIn();  
 })  
 return false;

После прохождения проверок на клиентской и серверных частях приложения, будет произведена попытка зарегистрировать пользователя:

final BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder(ENCRYPTION\_STRENGTH);  
  
final User user = new User();  
user.setUserName(username);  
user.setPassword(bCryptPasswordEncoder.encode(password));  
user.setIsAccountNonLocked(1);  
user.setRole("ROLE\_CUSTOMER");  
  
jdbcUserDao.addNewUserToDb(user);

В базе данных будут хранится зашифрованные пароли, для обеспечения большей безопасности. Шифровка происходит с помощью BcryptPasswordEncoder с силой шифрования 12. Всем новым пользователям назначается роль покупателя. Изначально пользователь разблокирован.

При добавлении уже существующего пользователя в СУБД, будет отправлено sql-исключение, которое обрабатывается следующим образом:

catch (final Exception e)  
{  
 model.addAttribute("error", true);  
 return PhoneshopPages.UserPages.*RegisterPage*;  
}

После обработки исключения, пользователь увидит ошибку об уже существующих данных пользователя. В случае успешной регистрации, пользователь будет перенаправлен на страницу для авторизации с сообщением об успешной регистрации:

model.addAttribute("isRegistered", true);

return PhoneshopPages.UserPages.LoginPage;

**4.3 Добавление устройства в базу данных**

Пользователь, авторизованный как администратор, может проводить различные операции над устройствами, хранящимися в системе, например добавление устройства.

Процесс добавления устройства начинается с ввода необходимой информации на соответствующей странице. Клиент отправляет запрос с данными устройства, которые должны быть добавлены в систему. Обработка запроса на сервере начинается с валидации:

phoneAddDtoValidator.validate(phoneAddDto, bindingResult);

Все поля проходят проверку на пустоту. Численные значения проходят проверку на значение. Особая валидация используется при валидации строки, указывающей на цвета устройства:

private boolean validateColorWithRegex(final String parameter)  
{  
 Pattern pattern = Pattern.compile("^\\d[\\d,]+?\\d$", Pattern.CASE\_INSENSITIVE);  
 Matcher matcher = pattern.matcher(parameter);  
 return matcher.find();  
}

Строка должна соответствовать виду “1001,1002,1003”. Соответствие введённой пользователем строки с заданным требованием проверяется с помощью регулярного выражения. При наличии ошибок после валидации пользователь увидит ту же страницу, но с уже ранее введёнными им данными и ему будет предложено изменить неправильно заданные значения.

if (bindingResult.hasErrors())  
{  
 model.addAttribute("recentData", phoneAddDto);  
 model.addAttribute("errors", bindingResult);  
 model.addAttribute("error", messageSource.getMessage("page.admin.error.adding.to.db", null, LocaleContextHolder.*getLocale*()));  
 return PhoneshopPages.UserPages.*AddDevicePage*;  
}

Если валидация прошла успешно, из объекта запроса создастся объект модели устройства. Далее формируется запрос в базу данных на добавление устройства:

jdbcPhoneDao.save(phone);

После сохранения устройства в базе данных обновляется информация об его наличии:

jdbcStockDao.update(savedPhone.getId(), quantity, 0L);

В конце сохранения устройства, заполняется таблица phone2color, связывающее устройство с доступными ему цветами:

for (final String color : colors)  
{  
 jdbcTemplate.update("insert into phone2color(phoneId,colorId) values (?,?)", savedPhone.getId(), color);  
}

При успешном завершении добавления устройства в систему, администратор увидит сообщение, подтверждающее успех операции.

**4.4 Обновление корзины**

При совершении покупок пользователь может редактировать корзину с устройствами. Возможно удаление устройства и обновление корзины (например, если покупатель хочет добавить некоторое количество элементов).

Для обновления корзины пользователь должен перейти на страницу корзины, изменить данные в необходимых полях, нажать кнопку. Процесс обновления корзины выглядит следующим образом:

final Cart cart = cartService.getCart(httpSession);  
  
if (cart.getCartItems().isEmpty())  
{  
 return prepareModelForEmptyCart(cart, model);  
}

Для проверки берётся текущая сессионная корзина пользователя. Проверка идёт на наличие элементов в корзине. Если корзина пустая, управление передаётся методу prepareModelForEmptyCart, который настраивает ответ с вложенным в него сообщением об ошибке. Если корзина не пустая происходит валидация полей запроса:

phoneArrayDtoValidator.validate(phoneArrayDto,

bindingResult);

Вовремя валидации идёт проход по списку устройств, которые должны быть обновлены, проверяется наличие устройства каждого устройства в базе данных, а также введённые числовые значения. Количество введённое в запросе не должно быть равно нулю.

PhoneArrayDto dto = (PhoneArrayDto) o;  
for (int i = 0; i < dto.getQuantity().length; i++)  
{  
 try  
 {  
 final Stock stock = jdbcStockDao.get(Long.*parseLong*(dto.getPhoneId()[i])).orElse(null);  
 final long quantity = Long.*parseLong*(dto.getQuantity()[i]);  
 if (quantity <= 0 || stock == null || quantity > (stock.getStock() - stock.getReserved()))  
 {  
 errors.rejectValue("quantity", dto.getPhoneId()[i]);  
 }  
 }  
 catch (NumberFormatException e)  
 {  
 errors.rejectValue("quantity", dto.getPhoneId()[i]);  
 }  
}

После валидации, если возникли ошибки, ошибки заполняются в модель, пользователь увидит страницу с сообщением об ошибке. Если валидация прошла успешно, создаётся коллекция типа ключ-значение, куда сохраняются значения идентификатора устройства и количества для обновления.

final HashMap<Long, Long> idQuan = new HashMap<>();  
for (int i = 0; i < getQuantity().length; i++)  
{  
 idQuan.put(Long.parseLong(phoneArrayDto.getPhoneId()[i]),  
 Long.parseLong(phoneArrayDto.getQuantity()[i]));  
}

Сформированная коллекция передаётся в метод-сервис для обновления корзины. Изначально метод находит существующие элементы корзины, получает их значения. После этого, для каждого элемента корзины, который должен обновиться проверяется возможность добавления такого количества элементов (из-за ограничений в наличии). После успешного прохождения проверки, количество устройств в элементе корзины изменяется, иначе остаётся неизменным. После обновления элементов корзины она пересчитывается, устанавливая новую общую цену и количество.

items.keySet().stream()

.map(phoneId -> findCartItem(phoneId, cart))  
 .filter(Optional::isPresent)  
 .map(Optional::get)  
 .forEach(cartItem ->  
 {  
 final Long phoneId = cartItem.getPhone().getId();  
 final Long quantity = items.get(cartItem.getPhone().getId());  
 final Long quantityDifference = quantity - cartItem.getQuantity();  
 if (checkQuantity(phoneId, quantityDifference))  
 {  
 cartItem.setQuantity(cartItem.getQuantity() + quantityDifference);  
 }  
 });  
calculateCart(cart);

После успешного обновления корзины, пользователь увидит сообщение об успехе, если возникли ошибки, они будут указаны около поля.

**4.5 Поиск и сортировка устройств**

Для удобства пользователей в приложении реализована сортировка элементов по стоимости, по бренду, по модели и по размеру экрана. Также в приложении присутствует поиск элементов по названию модели.

Чтобы осуществить поиск или сортировку пользователь должен находиться на главной странице приложения. При отправке запроса на отображение этой страницы в строке запроса браузера передаются опциональные параметры: search, field, order, page. Они являются опциональными, так как поиск или сортировка не всегда запрашиваются пользователем. Когда пользователь задал в поисковое поле какое-либо название устройства или нажал любую кнопку сортировки и оно передалось в обработчик запросов инициализируется объект класса ParamsForSearch:

final ParamsForSearch paramsForSearch = new ParamsForSearch(search,  
 field, order, (int) offset,  
 QUANTITY\_ON\_PAGE.intValue());

Объект этого класса используется для обобщения параметров поиска и сортировки и используется для установки принципа получения устройств из базы данных. Изначально, при поиске устройств в базе данных, значения из объекта устанавливаются в отдельные переменные, для упрощённого доступа к ним и формируется изначальный sql-запрос:

final String search = paramsForSearch.getSearch();

final String sortField = paramsForSearch.getSortField();  
final String order = paramsForSearch.getOrder();  
final int offset = paramsForSearch.getOffset();  
final int limit = paramsForSearch.getLimit();  
final List<Object> objects = new ArrayList<>();  
final List<Integer> types = new ArrayList<>();

String query = SQL\_GET\_ALL\_PHONES + SQL\_WHERE\_SEARCH;

После этого производится проверка поля search на пустоту. Если поле не пустое, то параметр дополняет query (sql-запрос). Теперь будет происходить поиск элементов с учётом названия модели:

if (search != null && !search.isEmpty())  
{  
 query = query + " and lower(model) like lower(?) ";  
 objects.add("%" + search + "%");  
 types.add(Types.*VARCHAR*);  
}

Далее производится проверка на пустоту параметров sortField и order. Оба параметра должны быть или пустые одновременно, либо оба заполнены. Если параметры не пусты, то строка запроса дополняется параметрами для сортирования по первому и второму параметру. Если параметры пусты, то сортировка производиться не будет.

if (sortField != null && order != null)  
{  
 query = query + String.format("group by phones.id,phone2color.colorId order by %s %s ", sortField, order);  
}

После этого запрос дополняется параметрами limit и offset, что позволяет отображать устройства с разделением на страницы. Выполняется sql-запрос. Результат полученный из после выполнения запроса упаковывается в список устройств, который будет отображаться на странице. Создаётся коллекция типа ключ-значение, в которой элементы не могут повторяться. Это служит для добавления нескольких цветов к одному уникальному устройству.

final Map<Phone, Set<Color>> data = new LinkedHashMap<>();

После этого идёт проход по циклу, в котором инициализируется объект устройства, этот объект помещается в коллекцию data:

final Phone phone = configurePhoneFields(resultSet);  
data.putIfAbsent(phone, new HashSet<>());

После этого цвет привязывается к телефону в коллекции:

final Color color = new Color();  
color.setId(resultSet.getLong("colors.id"));  
color.setCode(resultSet.getString("code"));  
data.get(phone).add(color);

Цикл по получению данных из ответа бд продолжается до его конца. После этого коллекция с данными преобразуется в список телефонов, который получит пользователь.

Далее высчитывается количество страниц, на которых будет отображаться список телефонов:

final Long phoneQuantity = phoneServiceImpl.countPhoneQuantity(paramsForSearch);  
  
long numOfPages = phoneQuantity / *QUANTITY\_ON\_PAGE*;

После этого производится вычисление последней страницы, на которой будут элементы списка:

long lastPage;

if (phoneQuantity % QUANTITY\_ON\_PAGE != 0)  
{  
 lastPage = numOfPages + 2;  
}  
else  
{  
 lastPage = numOfPages + 1;  
}

Все высчитанные данные передаются в ответе на клиентскую часть приложения.