Refactoring and Design Improvement of OnlineBookStore Java Web Application

Group 9:

2702241553 - Brian Casey Reynard

2702245034 - Dedrick Justin

2702262766 - Paisal Tanjung

2702279142 - Riccardo Rehyan Setiawan

UNIVERSITAS BINA NUSANTARA
JAKARTA
2025

BAB 1
INTRODUCTION

Proyek OnlineBookStore merupakan sebuah aplikasi web berbasis Java yang dirancang untuk memfasilitasi penjualan buku secara daring. Aplikasi ini dibangun dengan memanfaatkan teknologi *Java Servlet*, *JDBC*, *MySQL*, serta HTML, CSS, dan dijalankan pada server Tomcat dengan menggunakan Maven sebagai pengelola dependensi. Dalam aplikasi ini, pengguna dapat melakukan registrasi, melakukan login, menelusuri daftar buku, menambahkan buku ke dalam keranjang belanja, melakukan pembelian, serta menerima struk pembayaran sebagai bukti transaksi. Sementara itu, administrator memiliki kemampuan untuk mengelola data buku dengan menambahkan, menghapus, maupun memodifikasi informasi buku yang tersedia.

Meskipun secara fungsional aplikasi ini sudah berjalan dengan baik, namun setelah dilakukan peninjauan lebih mendalam, ditemukan beberapa permasalahan struktural pada kode programnya. Permasalahan tersebut di antaranya adalah adanya *code duplication* (duplikasi kode), *long methods* (metode yang terlalu panjang), pencampuran logika bisnis dan tampilan, serta pengelolaan sesi yang tersebar dan tidak terpusat. Kompleksitas dan penyebaran tanggung jawab yang tidak jelas ini mengakibatkan kesulitan dalam pengembangan dan pemeliharaan kode di masa depan. Oleh sebab itu, proyek ini sangat tepat dijadikan sebagai objek studi kasus untuk penerapan refaktorisasi dengan menggunakan prinsip-prinsip desain perangkat lunak modern.

BAB II.
METHODOLOGY

1. Analisis Struktur Awal Proyek

Secara umum, proyek OnlineBookStore disusun dalam beberapa direktori utama sebagai berikut:

- src/main/java
  Folder ini berisi seluruh kode sumber aplikasi, termasuk di dalamnya berbagai *class* servlet, *utility class*, dan file konfigurasi logika bisnis.
- WebContent
  Folder ini menyimpan seluruh file tampilan web, termasuk file HTML, CSS, JavaScript, serta file JSP jika digunakan.
- File Konfigurasi
  Beberapa file penting yang berkaitan dengan konfigurasi proyek disimpan di sini, di antaranya *pom.xml* untuk pengelolaan dependensi Maven, *Dockerfile* untuk keperluan kontainerisasi, serta *application.properties* yang memuat pengaturan koneksi ke database MySQL.

Struktur folder ini pada awalnya tampak teratur, namun setelah dilakukan penelusuran lebih mendalam, ditemukan bahwa logika bisnis, pengolahan data, serta tampilan seringkali bercampur di dalam *class* servlet, sehingga mengaburkan pembagian tanggung jawab kode secara jelas.

2. Identifikasi Code Smell

Proses identifikasi *code smell* dilakukan melalui peninjauan manual terhadap kode pada beberapa *class* kunci, baik pada servlet maupun *utility class*. Beberapa temuan utama yang berhasil diidentifikasi antara lain:

a. Duplicate Code
   - File: Semua file servlet (misalnya *AddBookServlet.java, CartServlet.java, ViewBookServlet.java*, dan lainnya).
   - Problem: Ditemukan adanya pengulangan kode yang sama seperti pengaturan *content type*, pemeriksaan sesi login, serta penanganan error di hampir semua servlet.

b. Long Method
   - File: *AddBookServlet.java, CartServlet.java, ViewBookServlet.java*.
   - Problem: *Method Service()* di masing-masing servlet memiliki panjang yang berlebihan dan memuat banyak logika sekaligus dalam satu tempat.

c. Feature Envy
   - File: *StoreUtil.java*.
   - Problem: *Method updateCartItems()* terlalu kompleks karena menangani berbagai jenis operasi sekaligus terkait keranjang belanja.

d. Inappropriate Intimacy
   - File: Seluruh *class* servlet.
   - Problem: Pengaksesan atribut sesi dilakukan secara langsung oleh masing-masing servlet, tanpa adanya pengelolaan terpusat.

e. God Class dan Divergent Change (Kurangnya Kohesi)
   - File: *ViewBookServlet.java, CartServlet.java*.
   - Problem: Dalam *class* tersebut ditemukan pencampuran tanggung jawab yang sangat beragam, mulai dari validasi sesi, pengambilan data, pembaruan keranjang, hingga pembuatan tampilan HTML. Hal ini melanggar prinsip *Single Responsibility*.

f. Excessive Comments
   - File: Seluruh file servlet.
   - Problem: Banyak komentar yang sebenarnya hanya menjelaskan kembali isi kode secara berlebihan dan tidak memberikan informasi tambahan.

3. Perancangan Solusi

Berdasarkan hasil identifikasi permasalahan, beberapa *design pattern* dan prinsip rekayasa perangkat lunak diterapkan dalam proses refaktorisasi, di antaranya:

- Template Method Pattern:
  Digunakan pada *BaseServlet* untuk menstandarisasi proses umum yang selalu dijalankan oleh semua servlet, seperti pengaturan *response* dan validasi sesi.
- Decomposition:
  Menerapkan pemecahan *method* panjang menjadi beberapa *method* kecil yang spesifik dan lebih mudah dipahami.
- Encapsulation:
  Pengelolaan atribut sesi dikonsolidasikan ke dalam *BaseServlet*, sehingga servlet lain tidak lagi melakukan akses langsung terhadap sesi.
- Single Responsibility Principle:
  Setiap *class* maupun *method* diberikan satu tanggung jawab utama yang jelas dan terfokus, sehingga meningkatkan kohesi dan mengurangi kompleksitas.

BAB III.
RESULT & ANALYSIS

Melalui proses refaktorisasi yang telah dilakukan, diperoleh berbagai peningkatan kualitas kode sebagai berikut:

- Penghapusan Duplicate Code:

  Dengan adanya BaseServlet, pengaturan awal servlet seperti pengaturan content type dan validasi sesi tidak lagi diulang-ulang di setiap servlet.

  Contoh *snipped code* dari AddBookServlet.java sebelum *refactoring*.

```
if (!StoreUtil.isLoggedIn(UserRole.SELLER,
req.getSession())) {
     RequestDispatcher rd =
     req.getRequestDispatcher("SellerLogin.html");
     rd.include(req, res);
     pw.println("<table  class=\"tab\"><tr><td>Please
     Login First to Continue!!</td></tr></table>");
     return;
}
```

  Contoh *snipped code* dari AddBookServlet.java setelah *refactoring*.

```
if (!StoreUtil.isLoggedIn(UserRole.SELLER,
req.getSession())) {
     handleNotLoggedIn(req, res, pw);
     return;
}
```

- Penyederhanaan Long Methods:

  Method yang sebelumnya terlalu panjang berhasil dipecah menjadi beberapa method kecil yang fungsional dan mudah diuji.

Contoh *snipped code* dari ViewBookServlet.java sebelum *refactoring*.

```
public void service(HttpServletRequest req,
HttpServletResponse res) throws IOException,
ServletException {
     // Check if the customer is logged in, or else return
     to login page
     try {
          // Read All available books from the database
          // Set Available Books tab as active
          // Show the heading for the page
          // Add or Remove items from the cart, if
          requested
          // Checkout Button
     } catch (Exception e) {
          e.printStackTrace();
     }
    }
```

Contoh *snipped code* dari ViewBookServlet.java setelah *refactoring*.

```
public void service(HttpServletRequest req,
HttpServletResponse res) throws IOException,
ServletException {
    }
protected void doBusinessLogic(HttpServletRequest req,
HttpServletResponse res, PrintWriter pw)
            throws ServletException, IOException {
     }
```

- Penyelesaian Feature Envy:

  Operasi pada keranjang belanja yang semula bercampur dalam satu method besar kini dipecah menjadi addToCart() dan removeFromCart() yang masing-masing fokus pada satu jenis operasi.

Contoh *snipped code* dari StoreUtil.java sebelum *refactoring*.

```
public static void updateCartItems(HttpServletRequest
req) {
     if(selectedBookId != null){
          if (req.getParameter("addToCart") != null) {
               // Add to cart
          }
          else {
               // Remove from cart
          }
     }
}
```

Contoh *snipped code* dari StoreUtil.java setelah *refactoring*.

```
public static void updateCartItems(HttpServletRequest
req) {
     if (req.getParameter("addToCart") != null) {
          // Call add to cart function
     } else if (req.getParameter("removeFromCart") !=
     null) {
          // Call remove to cart function
     }
}
private static void addToCart(HttpSession session,
String selectedBookId) {
     // Add To Cart
}
private static void removeFromCart(HttpSession session,
String selectedBookId) {
     // Remove From Cart
}
```

- Pengelolaan Sesi yang Lebih Baik:

  Dengan adanya pembungkus akses sesi di BaseServlet, semua servlet dapat melakukan pengambilan maupun penyimpanan atribut sesi secara konsisten dan terpusat.

Contoh *snipped code* dari CartServlet.java sebelum *refactoring*.

```
HttpSession session = req.getSession();
```

Contoh *snipped code* dari CartServlet.java setelah *refactoring*.

```
protected Object getSessionAttribute(HttpServletRequest
req, String name) {
        HttpSession session = req.getSession();
        return session.getAttribute(name);
    }
```

- Pengurangan God Class:

  Tanggung jawab dalam class servlet dipisahkan menjadi beberapa method fungsional, sehingga logika bisnis, logika tampilan, dan pengelolaan data tidak lagi bercampur.

  Contoh *snipped code* dari CartServlet.java sebelum *refactoring*.

```
public void service(HttpServletRequest req,
HttpServletResponse res) throws IOException,
ServletException {
     // Check if customer is logged in
     // Add/Remove Item from the cart if requested
     // store the comma separated bookIds of cart in the
     session
     // Read the books from the database with the
     respective bookIds
     // set cartItems and amountToPay in the session
}
```

  Contoh *snipped code* dari CartServlet.java setelah *refactoring*.

```
private void renderCartHeader(PrintWriter pw) {
}
private double renderCartItems(List<Book> books,
List<Cart> cartItems, HttpSession session, PrintWriter
pw) {
}
private String createTotalAmountRow(double amount) {
}
```

- Pembersihan Komentar Berlebihan:

  Kode menjadi lebih bersih karena komentar-komentar yang tidak relevan telah dihapus, sehingga pembaca kode dapat lebih fokus memahami alur logika dari nama method dan variabel yang telah cukup representatif.

BAB IV.
CONCLUSION

Proses refaktorisasi yang dilakukan terhadap proyek OnlineBookStore memberikan dampak positif yang signifikan terhadap keterbacaan, keterpeliharaan, dan kesiapan pengembangan lanjutan sistem. Dengan penerapan prinsip-prinsip desain perangkat lunak yang baik, kode sumber kini menjadi lebih modular, terstruktur, dan lebih mudah dikembangkan di masa depan. Setiap bagian kode memiliki tanggung jawab yang jelas, alur kerja yang konsisten, serta mudah untuk diuji, sehingga mengurangi potensi timbulnya kesalahan saat pengembangan lanjutan dilakukan. Laporan ini memberikan gambaran komprehensif mengenai proses refaktorisasi yang dilakukan, permasalahan awal yang dihadapi, strategi penyelesaiannya, serta hasil akhir perbaikannya.

REFERENCES

Fowler, M. (1999). Refactoring: Improving the Design of Existing Code. Addison-Wesley.

Suryanarayana, G., Samarthyam, G., & Sharma, T. (2014). Refactoring for Software Design Smells: Managing Technical Debt. Morgan Kaufmann.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Repository OnlineBookStore: https://github.com/bharah08/onlinebookstore-javaproject