**Contact Management System**

**Review Documentation**

**2310030084:P.Thanusree**

**2310030099:N.Vaishnavi**

**2310030106:K. Vaishnavi**

**2310030206:T.Charitha**

# Introduction

In Review 3 of the Contact Management System, the core focus shifts to establishing the **server-side infrastructure**, implementing backend logic using Java and **MongoDB**, and ensuring seamless connectivity between the server, database, and client-side interface. This

phase is essential in transforming the project from a static UI to a dynamic application that can handle real-time contact data operations like create, read, update, and delete (CRUD).

# Server Configuration

The server-side framework uses **Apache Tomcat 10**, which serves as the servlet container for running backend Java code.

# Apache Tomcat Setup

* + - Installed Apache Tomcat 10.
    - Configured Tomcat as a server within **Eclipse IDE**.
    - Ensured successful deployment via the [http://localhost:8080](http://localhost:8080/) address.
    - Added Tomcat runtime to the project’s build path.

# Project Structure

* + - Created a **Dynamic Web Project** called ContactManagement.
    - Typical structure:
      * src/ – Contains Java files including Servlets and MongoDB utility classes.
      * WebContent/ – HTML forms and JSPs for frontend interactions.
      * WEB-INF/ – Contains deployment descriptor web.xml.

# Servlet Mapping

Configured web.xml for routing requests: xml

CopyEdit

<servlet>

<servlet-name>AddContactServlet</servlet-name>

<servlet-class>com.contact.AddContactServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>AddContactServlet</servlet-name>

<url-pattern>/addContact</url-pattern>

</servlet-mapping>

# Backend Logic with MongoDB

MongoDB is used as the backend database, replacing traditional relational models with flexible document storage.

# MongoDB Configuration

* + - MongoDB installed and running on default port 27017.
    - Created database: contact\_db.
    - Collection used: contacts.

# Java-MongoDB Connectivity

Used the **MongoDB Java Driver** (org.mongodb:mongodb-driver-sync) to connect Java with MongoDB.

# MongoDB Utility Class

java CopyEdit

public class MongoUtil {

private static MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017");

public static MongoDatabase getDatabase() { return mongoClient.getDatabase("contact\_db");

}

public static MongoCollection<Document> getCollection() { return getDatabase().getCollection("contacts");

# Implementing Servlets for CRUD

The main operations are implemented using Java Servlets which communicate with MongoDB via the utility class.

# Adding a Contact

java CopyEdit

@WebServlet("/addContact")

public class AddContactServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String name = request.getParameter("name"); String email = request.getParameter("email"); String phone = request.getParameter("phone");

Document doc = new Document("name", name)

.append("email", email)

.append("phone", phone);

MongoUtil.getCollection().insertOne(doc); response.sendRedirect("success.jsp");

}

}

# Viewing Contacts

java CopyEdit

@WebServlet("/viewContacts")

public class ViewContactsServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

MongoCursor<Document> cursor = MongoUtil.getCollection().find().iterator(); request.setAttribute("cursor", cursor);

RequestDispatcher rd = request.getRequestDispatcher("view\_contacts.jsp"); rd.forward(request, response);

}

}

# Client-Server Integration

The frontend communicates with servlets using HTML forms. Each form is configured to send POST or GET requests to appropriate URLs mapped in the servlet configuration.

# Add Contact Form

html

<form method="post" action="addContact">

<label>Name:</label><input type="text" name="name" required><br>

<label>Email:</label><input type="email" name="email" required><br>

<label>Phone:</label><input type="text" name="phone" required><br>

<input type="submit" value="Add Contact">

</form>

# Response Handling

* + - Successful operations redirect to success.jsp.
    - In case of error, error.jsp displays error messages with debugging hints.

# MongoDB Collection Design

MongoDB collections are schema-less, which allows flexibility. However, we followed a consistent document structure for clarity and data validation.

# Sample Contact Document

json CopyEdit

{

"\_id": ObjectId("643dfc3f12d6a13b7e2b8d7a"), "name": "Alice Smith",

"email": "[alice@example.com](mailto:alice@example.com)",

"phone": "1234567890"

}

# Indexing

* + - MongoDB automatically indexes \_id.
    - Additional indexes (optional) can be added for faster search on email or phone.

# Debugging and Testing

* 1. **Test Cases**
     + **Form Submission Test**: Verify data inserted into MongoDB after form submission.
     + **Data Display Test**: View all contacts retrieved and listed properly.
     + **Error Simulation**: Turn off MongoDB and ensure proper exception handling.

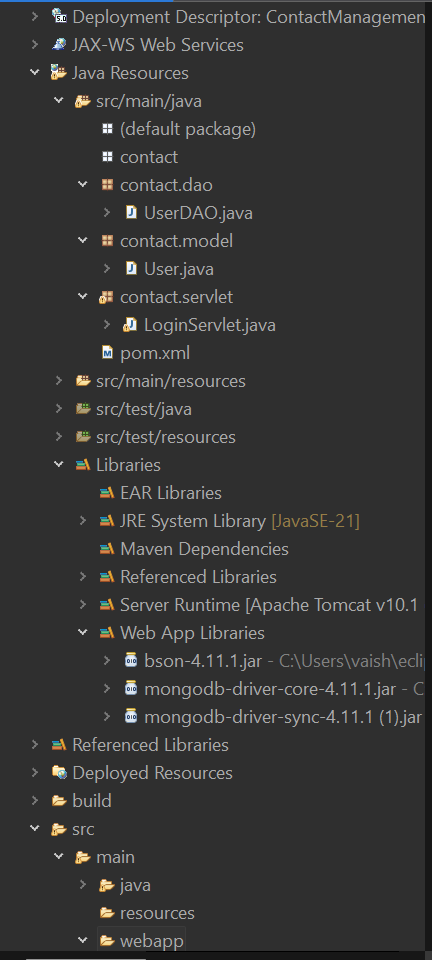
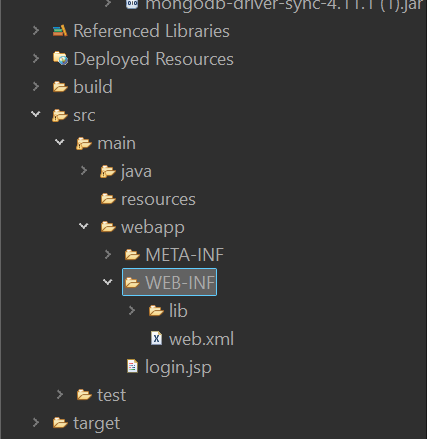
# Common Issues

* + - **Connection Error**: Fixed by correcting MongoDB URI in MongoUtil.
    - **NullPointerException**: Handled using form validation and try-catch blocks.
    - **Servlet Mapping Issue**: Resolved via web.xml review and annotation checks.

# Tools and Technologies Used

|  |  |
| --- | --- |
| **Component** | **Technology** |
| Server | Apache Tomcat 10 |
| Backend Language | Java (Servlets) |
| Database | MongoDB |
| IDE | Eclipse IDE |
| Build Tool | Maven (optional) |
| UI | HTML/CSS + JSP |

**8. Project Structure Overview**



ContactManagement/

├── src/

│ └── com.contact/

│ ├── AddContactServlet.java

│ ├── ViewContactsServlet.java

│ ├── MongoUtil.java

├── WebContent/

│ ├── add\_contact.html

│ ├── success.jsp

│ ├── error.jsp

│ └── view\_contacts.jsp

├── WEB-INF/

│ └── web.xml

# Conclusion

Review 3 demonstrates that the backend, server, and MongoDB components are working together effectively. The system is now capable of saving and managing contact data through real-time operations enabled by Java Servlets and MongoDB, with Apache Tomcat handling server-side execution. Seamless client-server communication ensures smooth CRUD functionality, laying a strong foundation for future development.This setup is both efficient and scalable, making it suitable for upcoming features such as editing, deleting, login functionality, and search capabilities. Additionally, the project can be enhanced with improved UI/UX design and migrated to the cloud using MongoDB Atlas for better accessibility and scalability.

THANK YOU