COS30041 Creating Secure and Scalable Software

Software Design Document for Distinction (D) Grade

Prepared by: Gia Bao Bui, 103533680

Tutor's name: Wei Lai

Demonstration to Tutor for Feedback: Week 12 Lab classes

Submission for Portfolio: Week 14, Monday, 9:00 am

Intended Learning Outcomes (extracted from Unit Outline)

1. Build and deploy secure and scalable application using contemporary frameworks
2. Explain and apply strategies, patterns and frameworks to address a range of scalability issues
3. Explain and apply strategies, patterns and frameworks to address a range of security issues
4. Use contemporary tools to evaluate the scalability and security of applications

Table of Contents

Introduction *3*

Business Scenario *3*

Software Requirements *3*

Requirements Justification 3

Functionality and Technology Matrix 4

Software Design *5*

Sample Coding *7*

Show all dogs functionality 7

Update user detail functionality 10

Submit an order functionality 13

Software Testing Result *16*

Show All Dogs Functionality Testing 16

Update User Detail Functionality Testing 17

Submit An Order Functionality Testing 20

Software Title: ED-DogsHaven

Introduction

A simplified E-commerce website to adopt dogs.

Business Scenario

ED-Dogs-Haven is a new business aiming to provide a safe online platform for people to buy their dogs. The web application follows the same logic as popular e-commerce platforms, in which administrators upload the dogs onto the platform and shoppers make orders to adopt those friends (Payment is not included in the scope of this application). Users and administrators must log in to perform their authorised actions on the system using their email. When logged in, shoppers can add items (dogs) to their cart and submit their orders. On the other hand, an administrator has the authorization to manage the whole website, in which they can update the products (dogs), view the orders and updates the shoppers' information.

Software Requirements

Requirements Justification

F1. Get all dogs from the database for the administrator. All dogs are listed (even the ones that have been sold) as there are possible changes to be made.
F2. Get all available dogs from the database for public access. Unavailable dogs are not retrieved.
F3. Add a dog for the administrator to add a new product to the platform.
F4. Update dog information for the administrator to modify the properties of the product on the platform.
F5. Show all orders for the administrator for business needs.
F6. Show orders of a user. This functionality is for users to keep track of their orders and cancel those if needed.
F7. Get the total price of the order. The total price will be calculated on the fly to avoid duplication of data in the database.
F8. Get the status of the order (whether the order has been resolved or not). The status of the order will be determined by whether every dog in the order has been sold (given to the new owner).
F9. Submit an order. All the dogs included in the order are marked as unavailable.
F10. Remove/Cancel an order. This functionality is for customers who do not wish to buy the product anymore. The dogs on the cancelled order will be available again. This functionality is also disabled if the order has been resolved.
F11. Add a dog to the cart when browsing the platform. The same dog cannot be in the same order.
F12. Remove a dog from the cart if a customer changes their mind.
F13. Register to be a customer of the platform. This functionality is available for public access. Everyone can be a customer.
F14. Show a customer detail. This functionality is available for both customers and administrators.
F15. Update customer detail. This functionality is available for both customers and administrators.
F16. Update the password. This functionality is available only for customers.
F17. Password validations. For a customer to update their detail, they have to enter their current password. For a customer to update their password, they have to enter their current password, a different new password and the same confirm password.
F18. Login. A customer and admin can access the system using a valid credential (email and password). To start shopping, a user must provide a valid customer account.
F19. Logout (terminating a user's session). Regarding customers, logout is automatically executed when they make changes to their email and password.
F20. Different displays and resources for different user groups (administrators/registered customers/unregistered users have different interfaces).

Functionality and Technology Matrix

The following table shows the relevant technologies discussed in this subject that could be used to implement the functionalities as suggested in the Functionalities section above.

Functionality	Related Technology discussed in this subject
F1	Web tier: Request managed bean (handles the request and calls the BLL session bean) Business tier: Stateless session bean (performs business logic, establishes the connection to the database, converts DogDAO to DogDTO and sends a collection back to the managed bean) **Regarding establishing the connection to the database and operations on the database, an Entity manager is used to perform those tasks. The entity classes are defined from the corresponding tables in the database with properties generated from their fields and relationships with other tables.*
F2	The same technologies as F1. Business logics are implemented to get only dogs that are available.
F3	The same as F1. However, the conversion is from DogDTO to DogDAO. Different operations on the database (INSERT INTO). A boolean result is sent back to the web tier.
F4	The same as F3. Different operations on the database (SET).
F5	Web tier: Request managed bean (handles the request and calls the BLL session bean) Business tier: Stateless session bean (performs business logic, establishes the connection to the database, converts OrderDAO to OrderDTO and sends a collection back to the managed bean)
F6	Web tier: Session managed bean (handles the customer session and calls the BLL session bean). A session bean is used to obtain the information of the user, and then create the appropriate interface. Business tier: same as F5
F7	Web tier: Request managed bean method with an OrderDTO parameter to calculate the individual price of each dog.
F8	Web tier: Request managed bean method with an OrderDTO parameter to calculate the individual price of each dog.
F9	Web tier: Session managed bean (handles the request and calls the BLL session bean). Session bean calls the Stateful session bean of a cart to perform shopping logic. When a user submits an order, the session bean already stored the user information and the current cart. Session bean calls the Stateless enterprise bean to perform the business logic. Business tier: Stateless session bean performs business logic, establishes the connection to the database, creates a new OrderDTO from the collection of DogDTO from the cart and the UserDTO, converts OrderDTO to OrderDTO and sends a Boolean indicating success or failure.
F10	Web tier: Session managed bean will handle the request as it contains UserDTO identity. Business Tier: Stateless session bean performs business logic, establishes the connection to the database, finds the existing OrderDAO and removes it.
F11	Web Tier: a session managed bean calls the stateful session bean cart.

	Business tier: stateful session bean cart handles the conversation with the user.
F12	The same as F11
F13	Web tier: Request managed bean handles the request and calls the BLL session bean. The managed bean sends new user information to the session bean. Business tier: Stateless session bean performs business logic, establishes the connection to the database, creates a new UserDTO from the data sent from the web tier, converts UserDTO to UserDAO and sends a Boolean indicating success or failure.
F14	Web tier: Session managed bean provides user key to generate their information. Business tier: Stateless session bean performs business logic, establishes the connection to the database, finds the corresponding UserDAO from the key, converts UserDAO to UserDTO and sends it back to the Web tier.
F15	Web tier: Session managed bean's properties are bound to form inputs. The submitted inputs are sent to the business logic layer. There is password validation at this stage to confirm the changes. When the update is success, log out the user to avoid data inconsistencies. Business tier: Stateless session bean performs business logic (validates the password), establishes the connection to the database, creates a new UserDTO from the properties, convert the UserDTO to UserDAO and merges that to the database.
F16	Web tier: calls Enterprise bean to validate the current password and ensure the new password is different from the old ones, validates the confirm password is the same as the new password. When validated, the session bean calls the enterprise bean to update the password. If the request is success, log out the user to avoid data inconsistencies. Business tier: Stateless session bean gets the old password with the user key provided from the web tier to validate password and sends corresponding Boolean value. The valid new password will be merged to the right user entity into the database.
F17	Described in F15, F16
F18	Web server: using JDBC realm to store user credentials based on the existing user table Web Tier: configurations made on login page and security constraints based on user groups for user to authenticate to access the right resources (web.xml). Stateful session bean handles the HTTP request to get the remote user and stores the key of the user to get user information throughout the session. Business tier: Because user uses email as the username when the primary key is userid, business tier handles this for easier queries throughout a user session.
F19	Web Tier: terminates a user session and reset all the properties.
F20	Web Tier: Session returns Boolean result if a user belongs to a specific group. Some UI components are working in conjunction with session bean to render the right information on the web page.

Software Design

1. I choose a 4-tier system for the software architecture.

EIS Tier: store data from the entity classes (handles data storage).

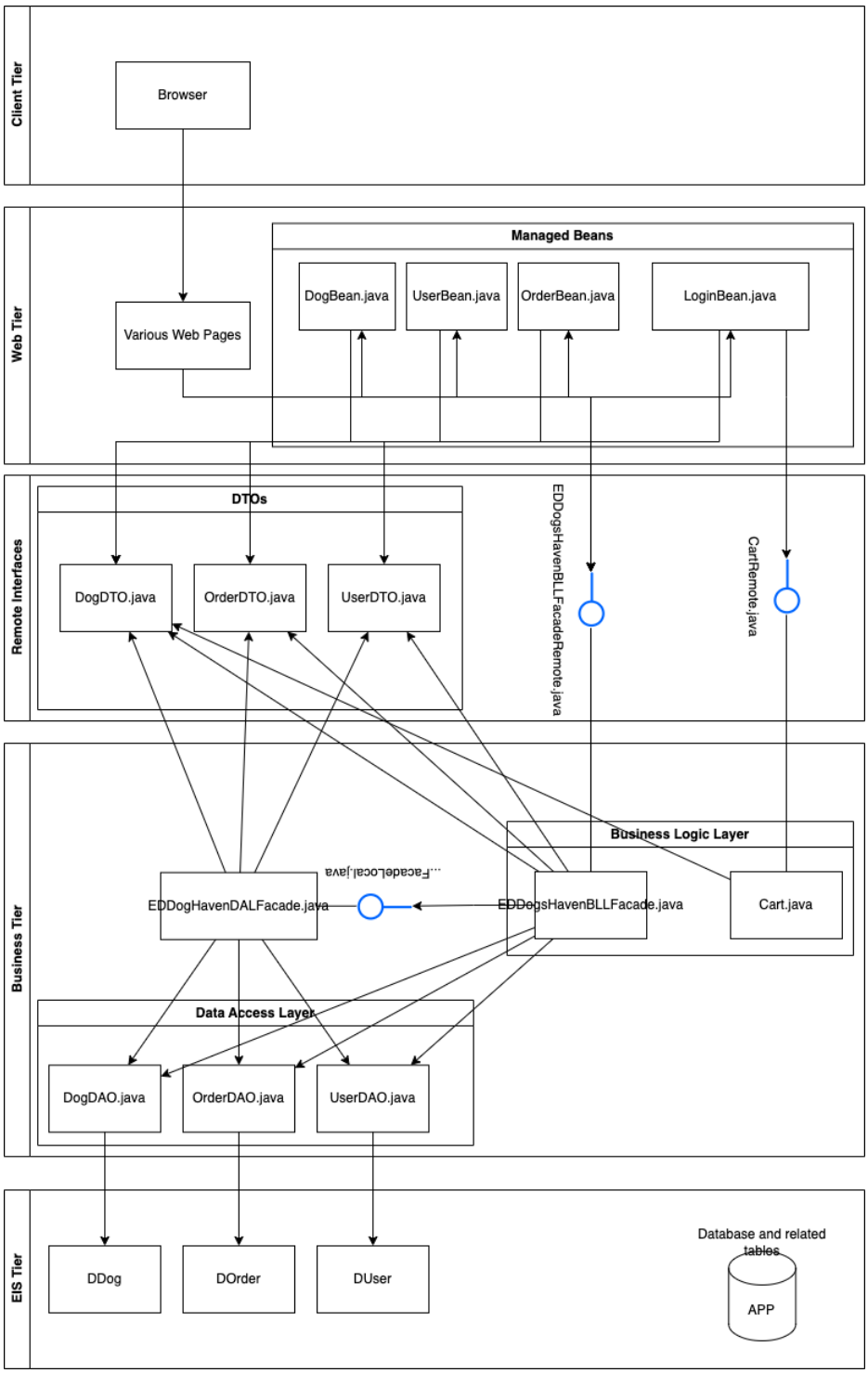
Business Tier: comprises of Business logic layer and Data access layer.

- Business Logic Layer: handles business logics of the enterprise application.
- Data Access Layer: consists of entity classes, connection establishment object to the database and ORM mechanism.

Web Tier: handles web presentation, website routing, data validation and conditional rendering.
Client Tier: browser.
2.

Tier	Components		Roles and responsibilities
EIS tier	Database tables		Store data of the business (Dog, Order, Order Item, User)
Business tier	Entity classes		Maps to the database tables (DogDAO, OrderDAO, UserDAO) Involves in operations on the database with Entity manager. Offers some pre-defined named queries.
	Session bean	Stateless session bean	Handles business logic (operations on Entity classes). Establishes database connections. ORMs: converts DAO to DTO.
		Stateful session bean	Holds long-running conversation with users (Keeps track of the Order items (Dog) in their virtual cart)
Web tier	Java Server Faces		Renders data from requests (including conditional rendering) Validates data. Handles user sessions (login/logout) Handles page outcome, routing.
Client tier	Browser		Sends HTTP requests to the web tier for resources.

3. I leveraged what I have been learning from the content of the unit and the tutorial. I tried to implement every tutorial feature into my final applications. In retrospect, I would have done things differently in some of the implementations. However, in general, the layers are logically separated from each other and it possibly scales well. The security feature is executed to the best practice, where a combination of email and password is usually preferred. Users are logged out when their information is intentionally modified.



Sample Coding

Show all dogs functionality

Dog Database

```
public void createDogTable() {
    Connection cnnct = null;
    Statement stmnt = null;
    try {
        cnnct = getConnection();
        stmnt = cnnct.createStatement();
        stmnt.execute("CREATE TABLE " + dogTable
                + " (DogId INT GENERATED ALWAYS AS IDENTITY, "
                + " Name CHAR(10), "
                + " Breed CHAR(30), "
                + " Age INT, "
                + " Characteristic CHAR(30), "
                + " Price DECIMAL(10,2), "
                + " IsAvailable BOOLEAN, "
                + " IsSold BOOLEAN, "
                + " Description VARCHAR(1000),"
                + " PRIMARY KEY (DogId))");
    } catch (SQLException ex) {
        while (ex != null) {
            ex.printStackTrace();
            ex = ex.getNextException();
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    } finally {
        if (stmnt != null) {
            try {
                stmnt.close();
            } catch (SQLException e) {
            }
        }
        if (cnnct != null) {
            try {
                cnnct.close();
            } catch (SQLException sqlEx) {
            }
        }
    }
}
```

DogDAO

```
/**
 *
 * @author giabaobui
 */
@Entity
@Table(name = "DDOG")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "DogDAO.findAll", query = "SELECT d FROM DogDAO d"),
    @NamedQuery(name = "DogDAO.findByDogid", query = "SELECT d FROM DogDAO d WHERE d.dogid = :dogid"),
    @NamedQuery(name = "DogDAO.findByName", query = "SELECT d FROM DogDAO d WHERE d.name = :name"),
    @NamedQuery(name = "DogDAO.findByBreed", query = "SELECT d FROM DogDAO d WHERE d.breed = :breed"),
    @NamedQuery(name = "DogDAO.findByAge", query = "SELECT d FROM DogDAO d WHERE d.age = :age"),
    @NamedQuery(name = "DogDAO.findByCharacteristic", query = "SELECT d FROM DogDAO d WHERE d.characteristic = :characteristic"),
    @NamedQuery(name = "DogDAO.findByPrice", query = "SELECT d FROM DogDAO d WHERE d.price = :price"),
    @NamedQuery(name = "DogDAO.findByIsavailable", query = "SELECT d FROM DogDAO d WHERE d.isavailable = :isavailable"),
    @NamedQuery(name = "DogDAO.findByIssold", query = "SELECT d FROM DogDAO d WHERE d.issold = :issold"),
    @NamedQuery(name = "DogDAO.findByDescription", query = "SELECT d FROM DogDAO d WHERE d.description = :description")})
public class DogDAO implements Serializable {

    @Size(max = 10)
```

Database query and ORM

```
248
249         @Override
            public Collection<DogDTO> getAllDogs() {
251             try {
252                 Query q = em.createNamedQuery("DogDAO.findAll");
253                 Collection<DogDAO> result = q.getResultList();
254                 return dogDAOCollectionToDogDTOCollection(result);
255             } catch (Exception e) {
                    e.printStackTrace();
257                 return null;
258             }
259         }
260
```

Business logic

```
        //used
        @RolesAllowed({"ED-Administrator"})
        @Override
        public Collection<DogDTO> getAllDogs() {
            try {
                return dal.getAllDogs();
            } catch (Exception e) {
                e.printStackTrace();
                return null;
            }
        }

        /**
         *
         * @return
         */
        //used
        @Override
        public Collection<DogDTO> getAllAvailableDogs() {
            try {
                Collection<DogDTO> allDogs = dal.getAllDogs();
                Collection<DogDTO> availableDogs = new ArrayList<>();
                for (DogDTO dog : allDogs) {
                    if (dog.getIsAvailable()) {
                        availableDogs.add(dog);
                    }
                }
                return availableDogs;
            } catch (Exception e) {
                e.printStackTrace();
                return null;
            }
        }
```

Managed bean methods

```
    public Collection<DogDTO> getAllDogsForCustomer() {
        try {
            return bll.getAllAvailableDogs();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public Collection<DogDTO> getAllDogsForAdmin() {
        try {
            return bll.getAllDogs();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

Data bindings

```
    <ui:repeat value="#{dogBean.allDogsForCustomer}" var="dog">
        <div style="border: solid 1px black; padding: 2%;">
            Dog ID:
            <h:outputText value="#{dog.dogId}"/>
            <br/>
            Name:
            <h:outputText value="#{dog.name}"/>
            <br/>
            Breed:
            <h:outputText value="#{dog.breed}"/>
            <br/>
            Characteristic:
            <h:outputText value="#{dog.characteristic}"/>
            <br/>
            Description:
            <h:outputText value="#{dog.description}"/>
            <br/>
            Age:
            <h:outputText value="#{dog.age}"/>
            <br/>
            Price:
            <h:outputText value="#{dog.price}"/>
            <br/>
            <h:form>
                <h:commandButton value="Add To Cart" action="#{loginBean.addCartItem(dog)}"/>
            </h:form>
        </div>
    </ui:repeat>
```

Update user detail functionality

```
    public void createUserTable() {
        Connection cnnct = null;
        Statement stmnt = null;
        try {
            cnnct = getConnection();
            stmnt = cnnct.createStatement();
            stmnt.execute("CREATE TABLE " + userTable
                    + " (UserId CHAR(10), "
                    + " Name CHAR(30), "
                    + " Password VARCHAR(30), "
                    + " Email VARCHAR(30) UNIQUE, "
                    + " Phone CHAR(10), "
                    + " Address CHAR(60), "
                    + " AppGroup VARCHAR(20), "
                    + " Active BOOLEAN, "
                    + " PRIMARY KEY (UserId))");
        } catch (SQLException ex) {
            while (ex != null) {
                ex.printStackTrace();
                ex = ex.getNextException();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            if (stmnt != null) {
                try {
                    stmnt.close();
                } catch (SQLException e) {
                }
            }
            if (cnnct != null) {
                try {
                    cnnct.close();
                } catch (SQLException sqlEx) {
                }
            }
        }
    }
```

```
/**
 *
 * @author giabaobui
 */
@Entity
@Table(name = "DUSER")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "UserDAO.findAll", query = "SELECT u FROM UserDAO u"),
    @NamedQuery(name = "UserDAO.findByUserid", query = "SELECT u FROM UserDAO u WHERE u.userid = :userid"),
    @NamedQuery(name = "UserDAO.findByName", query = "SELECT u FROM UserDAO u WHERE u.name = :name"),
    @NamedQuery(name = "UserDAO.findByPassword", query = "SELECT u FROM UserDAO u WHERE u.password = :password"),
    @NamedQuery(name = "UserDAO.findByEmail", query = "SELECT u FROM UserDAO u WHERE u.email = :email"),
    @NamedQuery(name = "UserDAO.findByPhone", query = "SELECT u FROM UserDAO u WHERE u.phone = :phone"),
    @NamedQuery(name = "UserDAO.findByAddress", query = "SELECT u FROM UserDAO u WHERE u.address = :address"),
    @NamedQuery(name = "UserDAO.findByAppgroup", query = "SELECT u FROM UserDAO u WHERE u.appgroup = :appgroup"),
    @NamedQuery(name = "UserDAO.findByActive", query = "SELECT u FROM UserDAO u WHERE u.active = :active")})
public class UserDAO implements Serializable {

    @Size(max = 30)
    @Column(name = "NAME")
```

```
114
115         @Override
            public void createAUser(UserDAO aUser) {
117             em.persist(aUser);
118         }
119
120         @Override
            public void updateAUser(UserDAO aUser) {
122             em.merge(aUser);
123         }
124
125         @Override
            public void removeAUser(UserDAO aUser) {
127             em.remove(em.merge(aUser));
128         }
129
130         @Override
            public UserDAO findAUser(Object aUserId) {
132             return em.find(UserDAO.class, aUserId);
133         }
134
135         @Override
            public UserDTO userDAOtoUserDTO(UserDAO aUser) {
137             return new UserDTO(
138                     aUser.getName(),
139                     aUser.getEmail(),
140                     aUser.getPhone(),
141                     aUser.getAddress(),
142                     aUser.getAppgroup(),
143                     aUser.getUserid(),
144                     aUser.getActive());
145         }
```

```
380         */
381         @RolesAllowed({"ED-Administrator", "ED-Customer"})
382         @Override
            public boolean updateAUser(UserDTO aUserDTO) {
384             try {
385                 UserDAO lAUserDAO = dal.findAUser(aUserDTO.getUserid());
386                 if (lAUserDAO != null) {
387                     // Update the user information
388                     lAUserDAO.setName(aUserDTO.getName());
389                     lAUserDAO.setEmail(aUserDTO.getEmail());
390                     lAUserDAO.setPhone(aUserDTO.getPhone());
391                     lAUserDAO.setAddress(aUserDTO.getAddress());
392                     // Perform the update operation
393                     dal.updateAUser(lAUserDAO);
394                     return true;
395                 }
396             } catch (Exception e) {
398                 e.printStackTrace();
399             }
400             return false;
401         }
```

Update user (for a customer's session).

```
public String updateUser() {
    try {
        if (username != null && userEmail != null && userPhone != null && userAddress != null) {
            UserDTO lUpdatedUser = new UserDTO(username, userEmail, userPhone, userAddress, userId);
            if (bll.updateAUser(lUpdatedUser)) {
                // Log out the user
                this.logout();
                return "success";
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return "error";
}
```

Update user (for an admin's session)

```
public String updateUser() {
    try {
        if (name != null && email != null && phone != null && address != null) {
            UserDTO lUpdatedUser = new UserDTO(name, email, phone, address, userId);
            if (bll.updateAUser(lUpdatedUser)) {
                return "success";
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return "failure";
}
```

```
                    <td>
                        <h:inputText value ="#{loginBean.userAddress}"
                                     title="Address" maxlength="30"
                                     required="false"/>
                    </td>
                </tr>
                <tr>
                    <td>
                        <h:outputLabel value="Email:"/>
                    </td>
                    <td>
                        <h:inputText value ="#{loginBean.userEmail}"
                                     title="Email" maxlength="30"
                                     required="false"/>
                    </td>
                </tr>
                <tr>
                    <td>
                        <h:outputLabel value="Password:"/>
                    </td>
                    <td>
                        <h:inputSecret id="password"
                                       title="Password"
                                       maxlength="8"
                                       required="true"
                                       requiredMessage="Please enter your password to proceed"
                                       value ="#{loginBean.oldPassword}"
                                       validator="#{loginBean.validateOldPassword}">
                        </h:inputSecret>
                        <h:message for="password" style="color:red"/>
                    </td>
                </tr>
            </tbody>
        </table>
        <p></p>
        <h:commandButton value="Update Details" action="#{loginBean.updateUser()}" />
    </h:form>
    <p></p>
    <p><h:link value="Back to All Customers" outcome="/customer/viewDetail.xhtml" /></p>
    <p><h:link value="Back to Dashboard" outcome="dashboard" /></p>
```

```
public void validateOldPassword(FacesContext context,
        UIComponent componentToValidate, Object value)
        throws ValidatorException {
    //get validate password
    oldPassword = (String) value;

    if (!bll.validatePassword(userId, oldPassword)) {
        FacesMessage message = new FacesMessage("Authentication failed! Please enter the right password");
        throw new ValidatorException(message);
    }
}
```

```
@Override
public boolean validatePassword(String aUserId, String aPassword) {
    try {
        UserDAO lUserDAO = dal.findAUser(aUserId);
        if (lUserDAO != null) {
            return (lUserDAO.getPassword().equals(aPassword));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}
```

Submit an order functionality

```
public void createOrderTable() {
    Connection cnnct = null;
    Statement stmnt = null;
    try {
        cnnct = getConnection();
        stmnt = cnnct.createStatement();
        stmnt.execute("CREATE TABLE " + orderTable
                + " (OrderId INT NOT NULL GENERATED ALWAYS AS IDENTITY, "
                + " CustomerId CHAR(10), "
                + " PRIMARY KEY (OrderId), "
                + " FOREIGN KEY (CustomerId) REFERENCES DUser(UserId))");
    } catch (SQLException ex) {
        while (ex != null) {
            ex.printStackTrace();
            ex = ex.getNextException();
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    } finally {
        if (stmnt != null) {
            try {
                stmnt.close();
            } catch (SQLException e) {
            }
        }
        if (cnnct != null) {
            try {
                cnnct.close();
            } catch (SQLException sqlEx) {
            }
        }
    }
}

public void createOrderItemTable() {
    Connection cnnct = null;
    Statement stmnt = null;
    try {
        cnnct = getConnection();
        stmnt = cnnct.createStatement();
        stmnt.execute("CREATE TABLE " + orderItemTable
                + " (OrderId INT NOT NULL, "
                + " DogId INT NOT NULL, "
                + " PRIMARY KEY (OrderId, DogId), "
```

```java
/**
 *
 * @author giabaobui
 */
@Entity
@Table(name = "DORDER")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "OrderDAO.findAll", query = "SELECT o FROM OrderDAO o"),
    @NamedQuery(name = "OrderDAO.findByOrderid", query = "SELECT o FROM OrderDAO o WHERE o.orderid = :orderid")})
public class OrderDAO implements Serializable {

    @JoinTable(name = "DORDERITEM", joinColumns = {
        @JoinColumn(name = "ORDERID", referencedColumnName = "ORDERID")}, inverseJoinColumns = {
        @JoinColumn(name = "DOGID", referencedColumnName = "DOGID")})
    @ManyToMany
    private Collection<DogDAO> dogDAOCollection;
    //primary key
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "ORDERID")
    private Integer orderid;
    @JoinColumn(name = "CUSTOMERID", referencedColumnName = "USERID")
    @ManyToOne
    private UserDAO customerid;
```

ORMs

```java
170         @Override
            public void createAnOrder(OrderDAO aOrder) {
172             em.persist(aOrder);
173         }
174
175         @Override
            public void updateAnOrder(OrderDAO aOrder) {
177             em.merge(aOrder);
178         }
179
180         @Override
            public void removeAnOrder(OrderDAO aOrder) {
182             em.remove(em.merge(aOrder));
183         }
184
185         @Override
            public OrderDAO findAnOrder(Object aOrderId) {
187             return em.find(OrderDAO.class, aOrderId);
188         }
189
190         @Override
            public OrderDTO orderDAOtoOrderDTO(OrderDAO aOrderDAO) {
192             return new OrderDTO(
193                     aOrderDAO.getOrderid(),
194                     this.userDAOtoUserDTO(aOrderDAO.getCustomerid()),
195                     this.dogDAOCollectionToDogDTOCollection(aOrderDAO.getDogDAOCollection()));
196         }
197
198         @Override
            public OrderDAO orderDTOToOrderDAO(OrderDTO aOrderDTO) {
200             //initialize a new user DAO
201             OrderDAO orderDAO = new OrderDAO();
202             //bind the user with their order
203             Object lAUserId = aOrderDTO.getCustomerid().getUserid();
204             orderDAO.setCustomerid(findAUser(lAUserId));
205             //set the items to the order
206             orderDAO.setDogDAOCollection(dogDTOCollectionToDogDAOCollection(aOrderDTO.getDogDTOCollection()));
207
208             return orderDAO;
209         }
210
```

Business logic

```
     */
    @RolesAllowed({"ED-Customer"})
    @Override
    public boolean submitAnOrder(Collection<DogDTO> aItems, Object aUserId) {
        try {
            //if the order has items
            if (!aItems.isEmpty() && aUserId != null) {
                //create a new order
                OrderDAO lNewOrder = new OrderDAO();
                //set the buyer to the current user (assuming the user has logged in the system)
                UserDAO lAUser = dal.findAUser(aUserId);
                lNewOrder.setCustomerid(lAUser);
                //set the collections of DogDTO -> DogDAO
                Collection<DogDAO> lItems = new ArrayList<>();
                for (DogDTO item : aItems) {
                    //get the dogs from the list
                    DogDAO lDog = dal.dogDTOToDogDAO(item);
                    //set the dogs to be temporarily unavailable (waiting to be actually adopted)
                    lDog.setIsavailable(Boolean.FALSE);
                    //update it to the database
                    dal.updateADog(lDog);
                    //add the dog to the order
                    lItems.add(lDog);
                }
                //set the items for the order
                lNewOrder.setDogDAOCollection(lItems);
                //persist the order into the database
                dal.createAnOrder(lNewOrder);
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }
```

```
@Stateful
public class Cart implements CartRemote {

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")
    private Collection<DogDTO> cart;

    public Cart() {
        cart = new ArrayList<>();
    }

    @Override
    public boolean addItem(DogDTO aDog) {
        try {
            if (aDog != null) {
                if (cart.isEmpty()) {
                    return cart.add(aDog);
                } else {
                    for (DogDTO dog : cart) {
                        if (aDog.getDogId().equals(dog.getDogId())) {
                            break;
                        }
                        return cart.add(aDog);
                    }
                }
            }
            return false;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
```

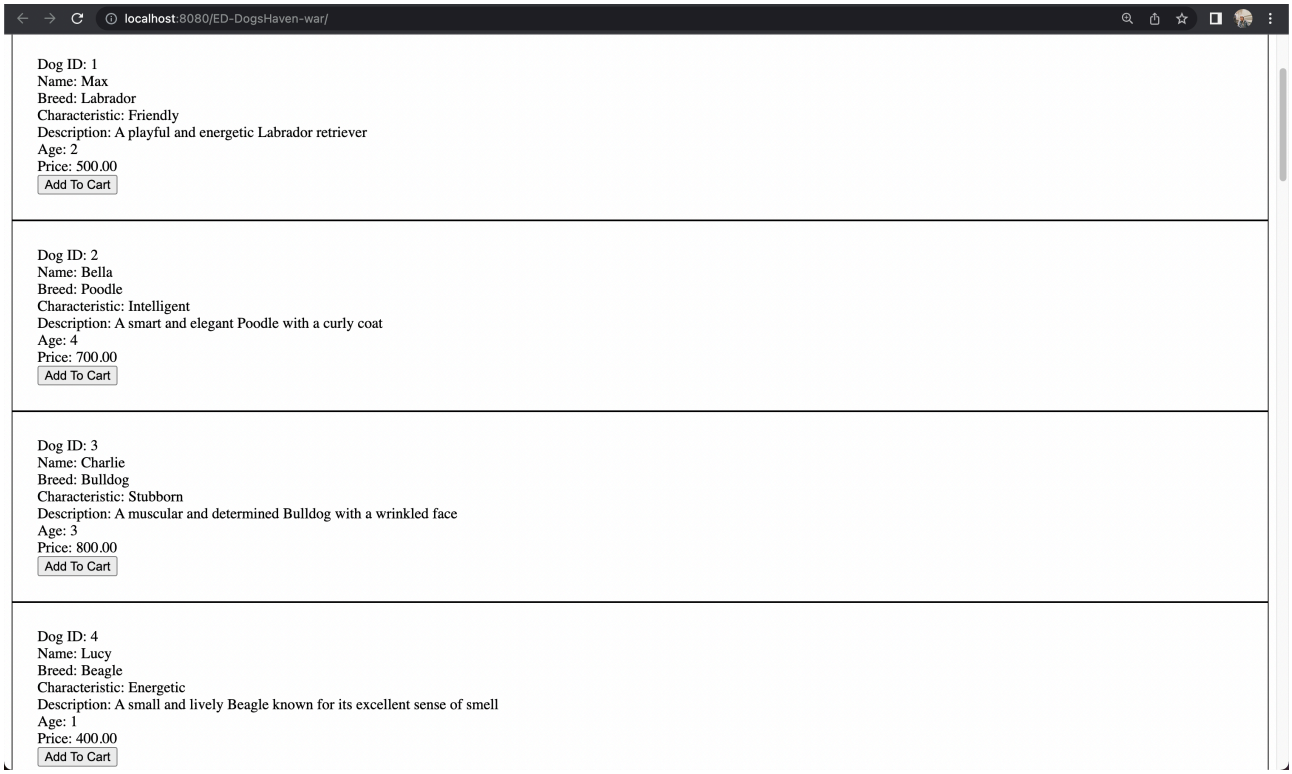
Managed Bean (calls a stateless and a stateful session bean)

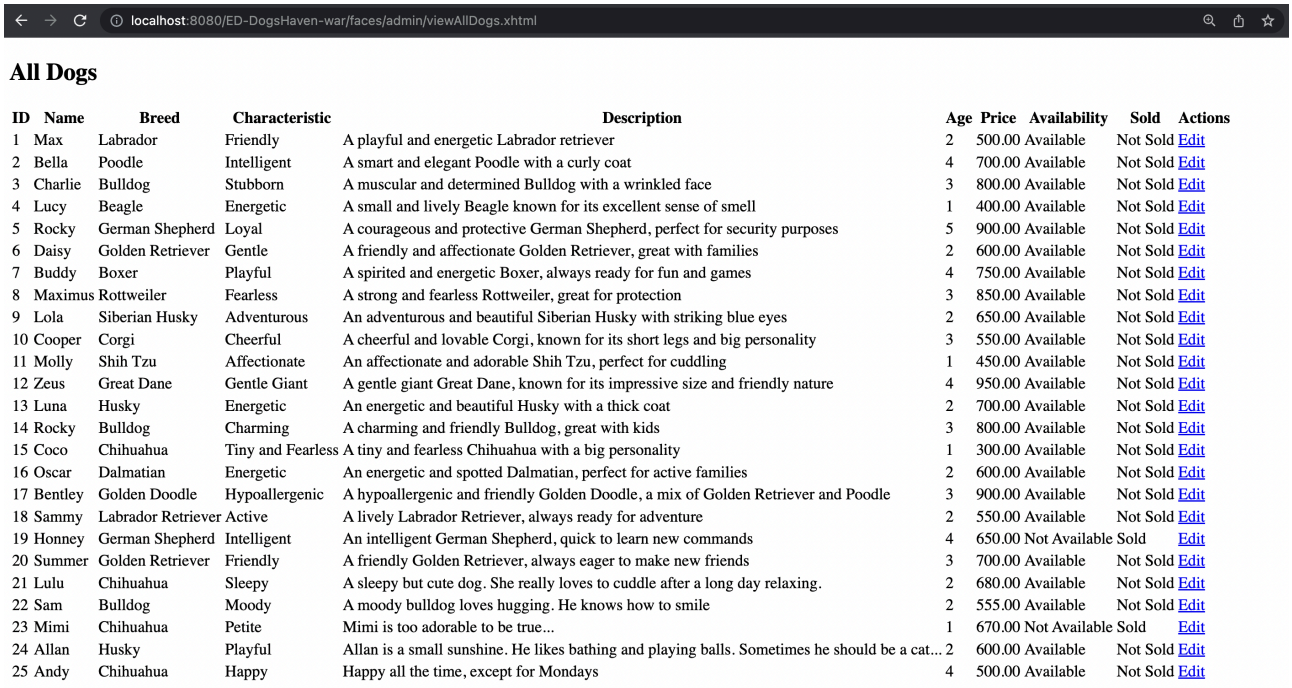
```java
public String addAnOrder() {
    try {
        if (isACustomer()) {
            if (bll.submitAnOrder(cart.getCart(), userId)) {
                cart.emptyCart();
                return "checkout-success";
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return "error";
}
```

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transiti
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
    <head>
        <title>Your Cart</title>
    </head>
    <body>
        <h1>Cart</h1>
        <h:form>
            <h:dataTable value="#{loginBean.cartItems}" var="dog">
                <h:column>
                    <f:facet name="header">Name</f:facet>
                        #{dog.name}
                </h:column>
                <h:column>
                    <f:facet name="header">Price</f:facet>
                        #{dog.price}
                </h:column>
                <h:column>
                    <h:form>
                        <h:commandButton value="Remove" action="#{loginBean.removeCartItem(dog.dogId)}"/>
                    </h:form>
                </h:column>
            </h:dataTable>
            <h3>Total Price: #{loginBean.totalPrice}</h3>
            <h:commandButton value="Checkout" action="#{loginBean.addAnOrder()}" />
        </h:form>
        <div>Back to <a href="/ED-DogsHaven-war/faces/main.xhtml">Main</a> to continue shopping</div>
    </body>
</html>
```

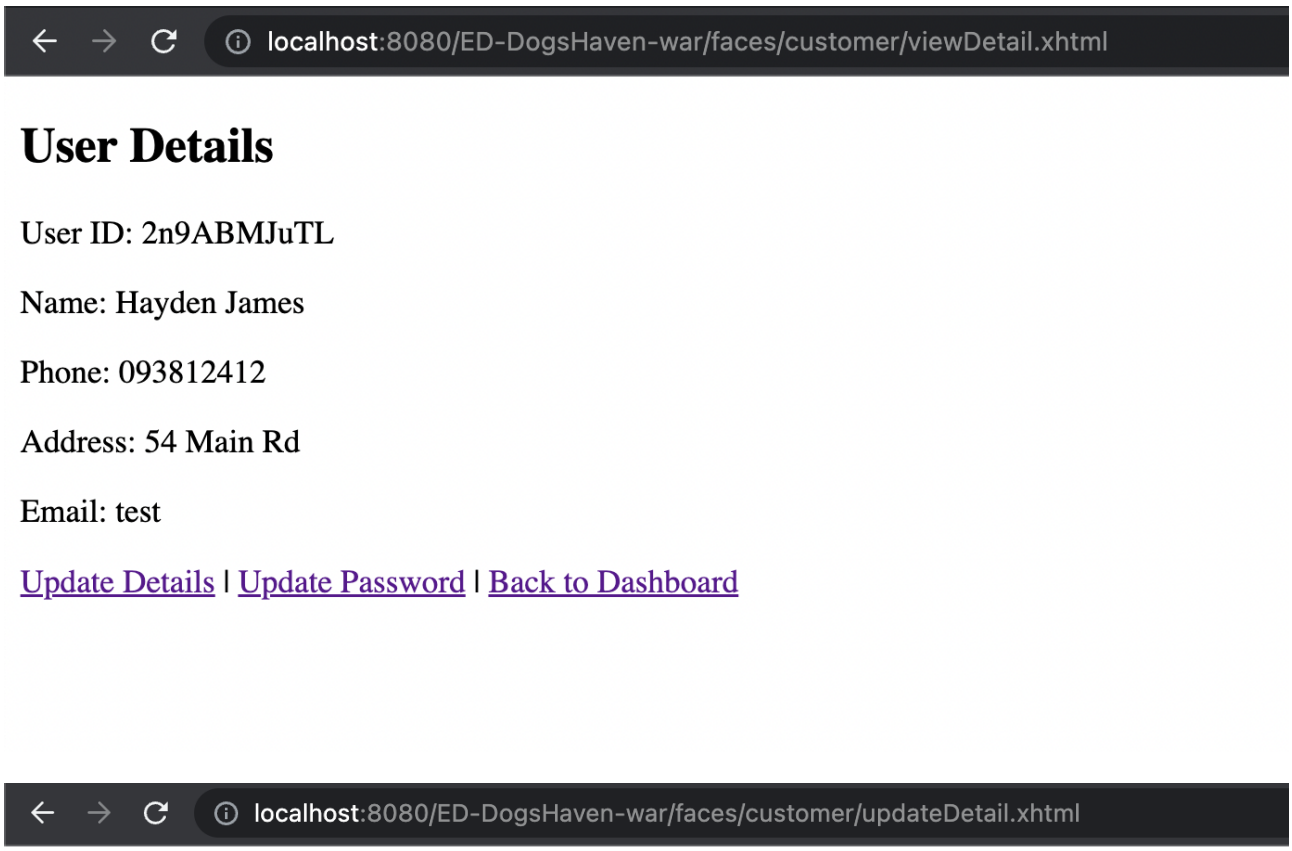
Software Testing Result

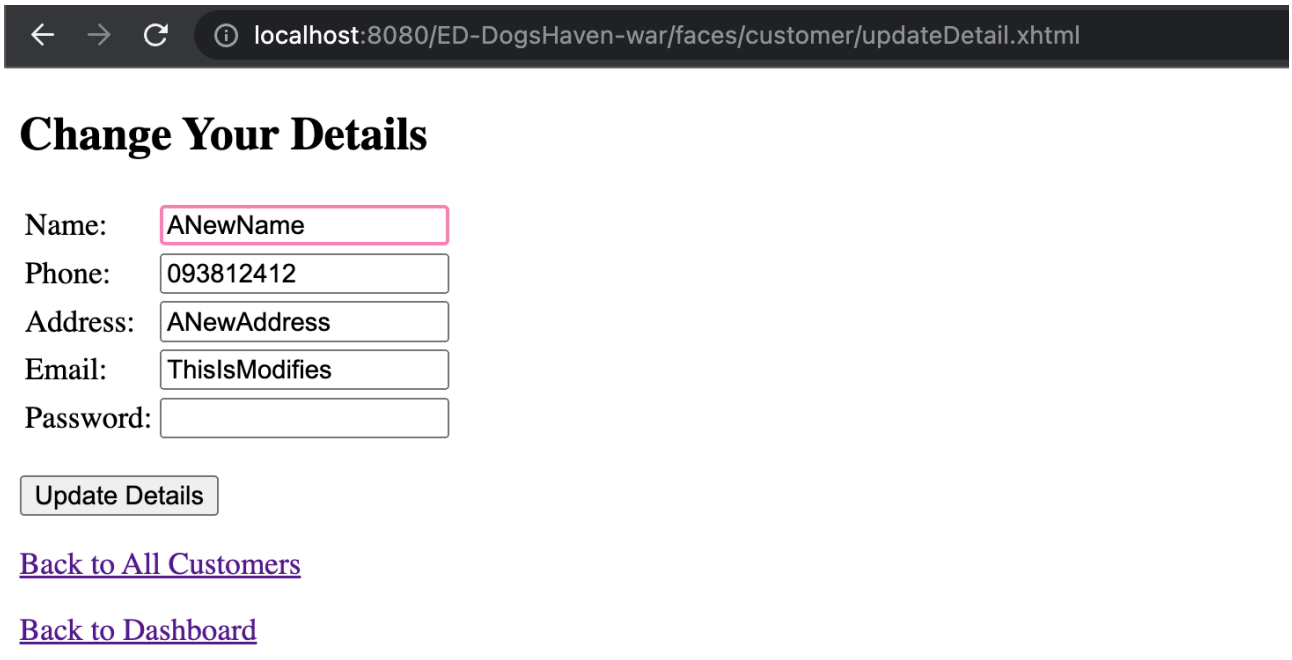
Show All Dogs Functionality Testing





Update User Detail Functionality Testing


localhost:8080/ED-DogsHaven-war/faces/customer/viewDetail.xhtml
User Details
User ID: 2n9ABMJuTL
Name: Hayden James
Phone: 093812412
Address: 54 Main Rd
Email: test
Update Details | Update Password | Back to Dashboard


localhost:8080/ED-DogsHaven-war/faces/customer/updateDetail.xhtml
Change Your Details
Name: ANewName
Phone: 093812412
Address: ANewAddress
Email: ThisIsModifies
Password:
Update Details
Back to All Customers
Back to Dashboard

Change Your Details

Name: ANewName
Phone: 093812412
Address: ANewAddress
Email: ThisIsModifies
Password: Authentication failed! Please enter the right password

Update Details

Back to All Customers

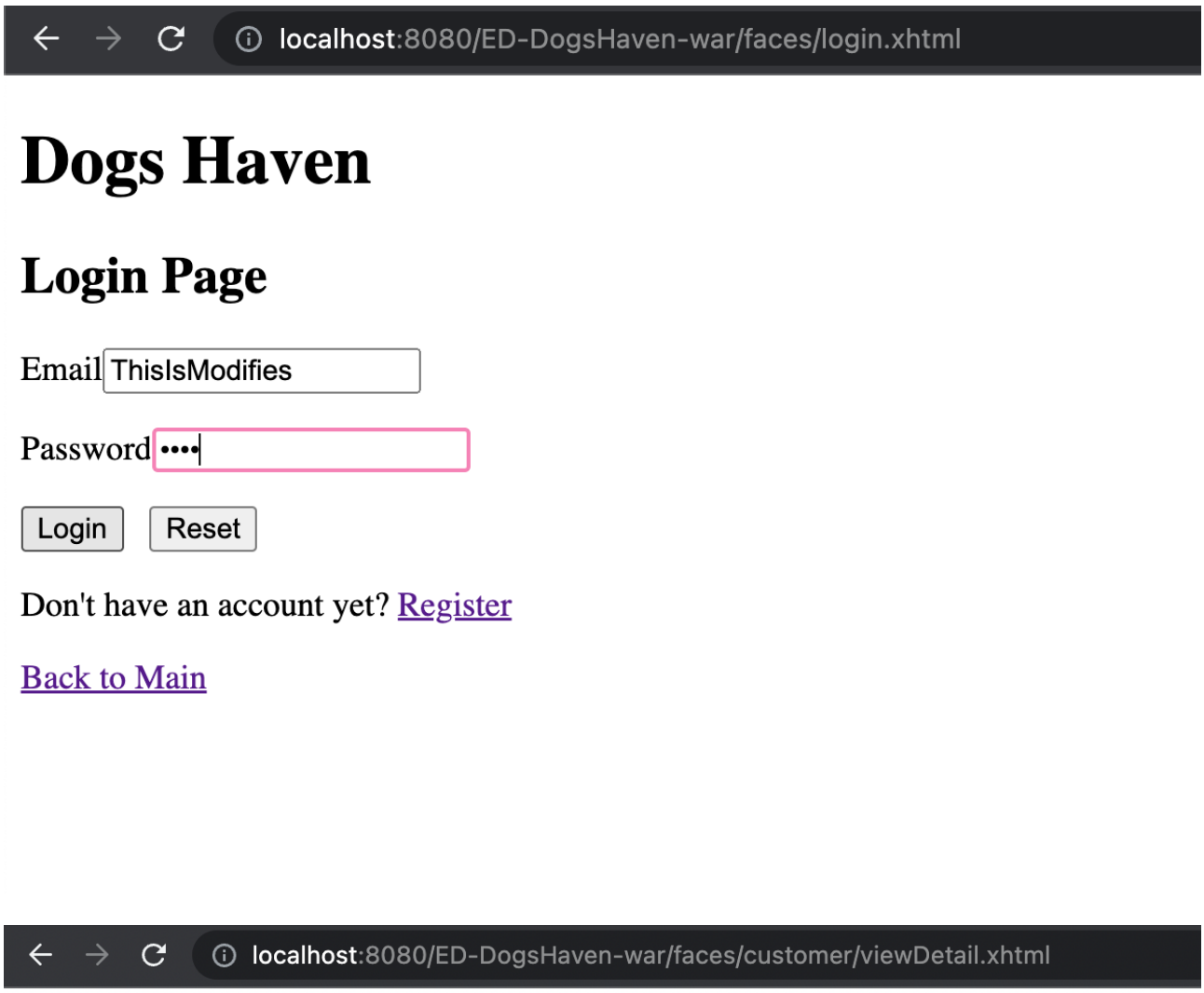
Back to Dashboard

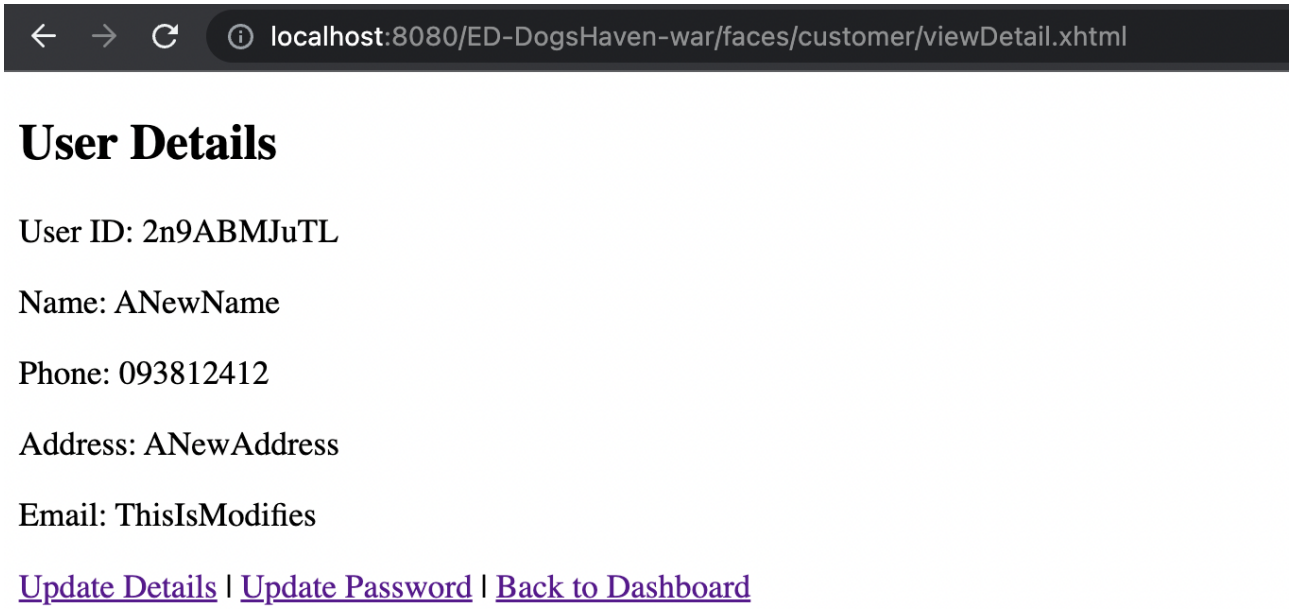
Update Details Success

Your details have been successfully updated.

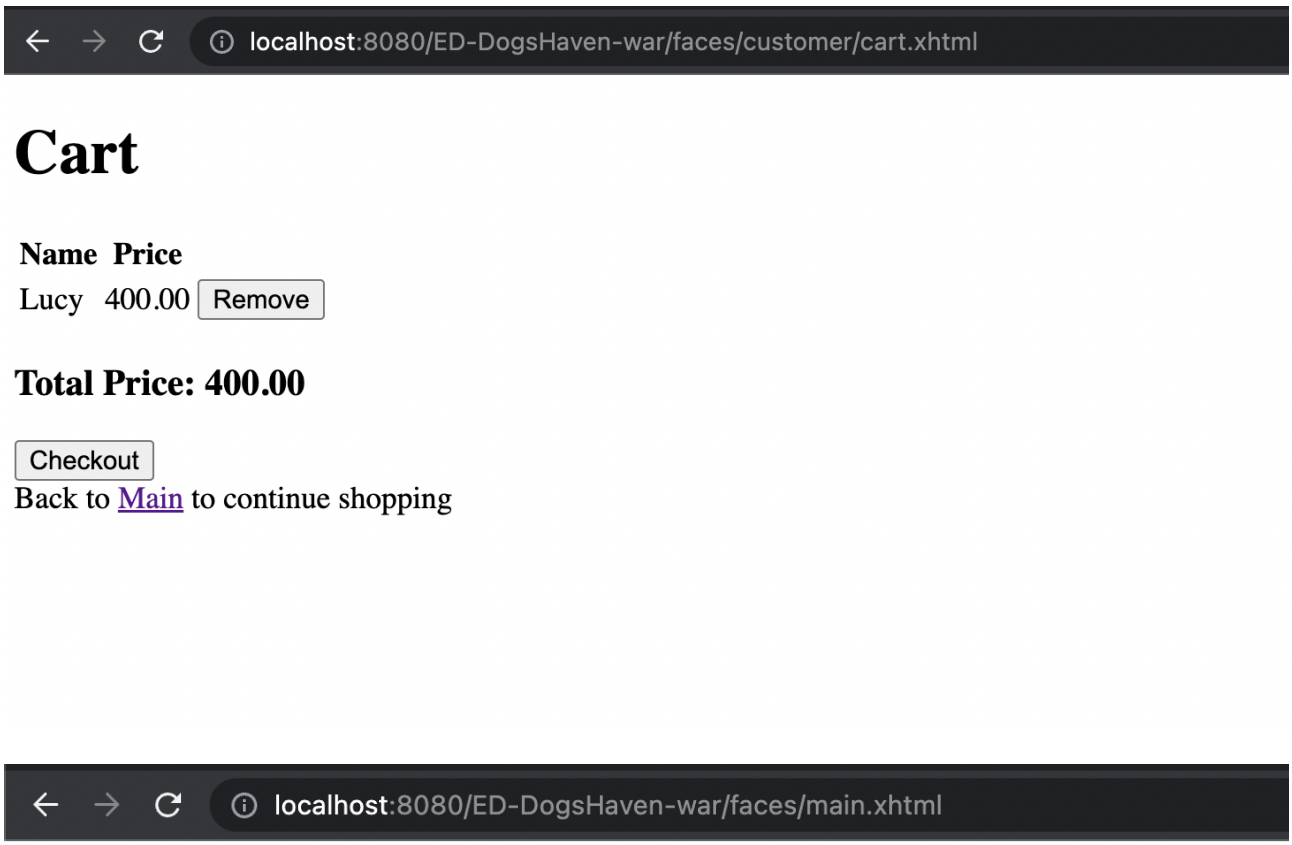
You are now logged out.

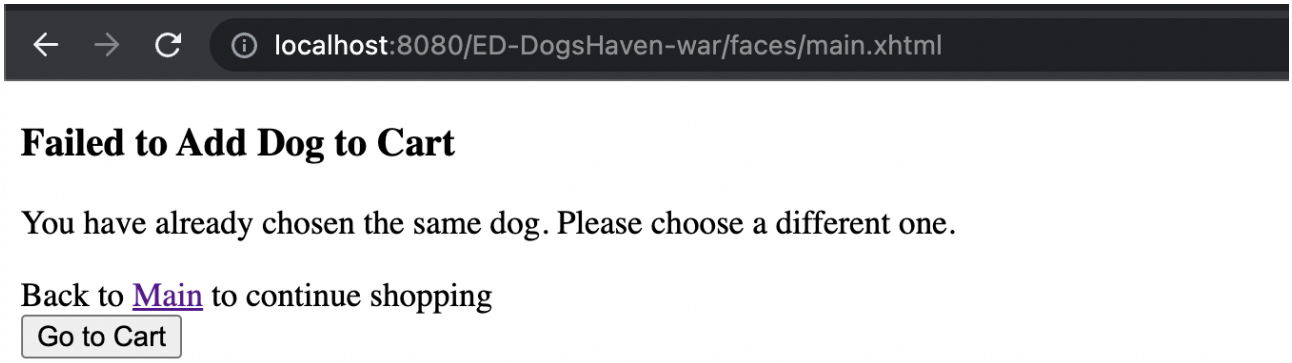
Back to Main



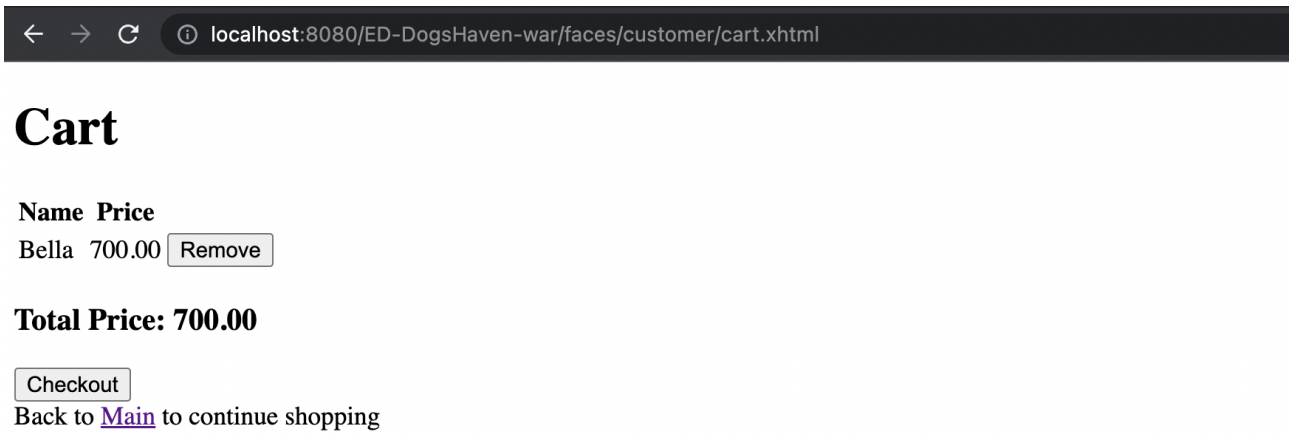


Submit An Order Functionality Testing


localhost:8080/ED-DogsHaven-war/faces/customer/cart.xhtml
Cart
Name Price
Lucy 400.00 Remove
Total Price: 400.00
Checkout
Back to Main to continue shopping


localhost:8080/ED-DogsHaven-war/faces/main.xhtml
Failed to Add Dog to Cart
You have already chosen the same dog. Please choose a different one.
Back to Main to continue shopping
Go to Cart


localhost:8080/ED-DogsHaven-war/faces/customer/cart.xhtml
Cart
Name Price
Lucy 400.00 Remove
Bella 700.00 Remove
Total Price: 1100.00
Checkout
Back to Main to continue shopping


localhost:8080/ED-DogsHaven-war/faces/customer/cart.xhtml
Cart
Name Price
Bella 700.00 Remove
Total Price: 700.00
Checkout
Back to Main to continue shopping

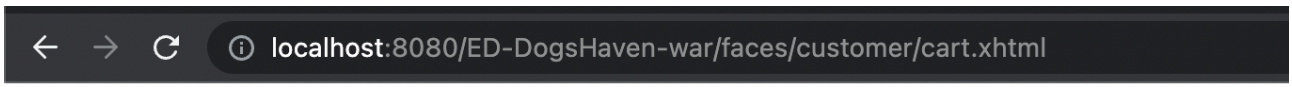
Cart

Name	Price	
Bella	700.00	Remove
Cooper	550.00	Remove

Total Price: 1250.00

Checkout

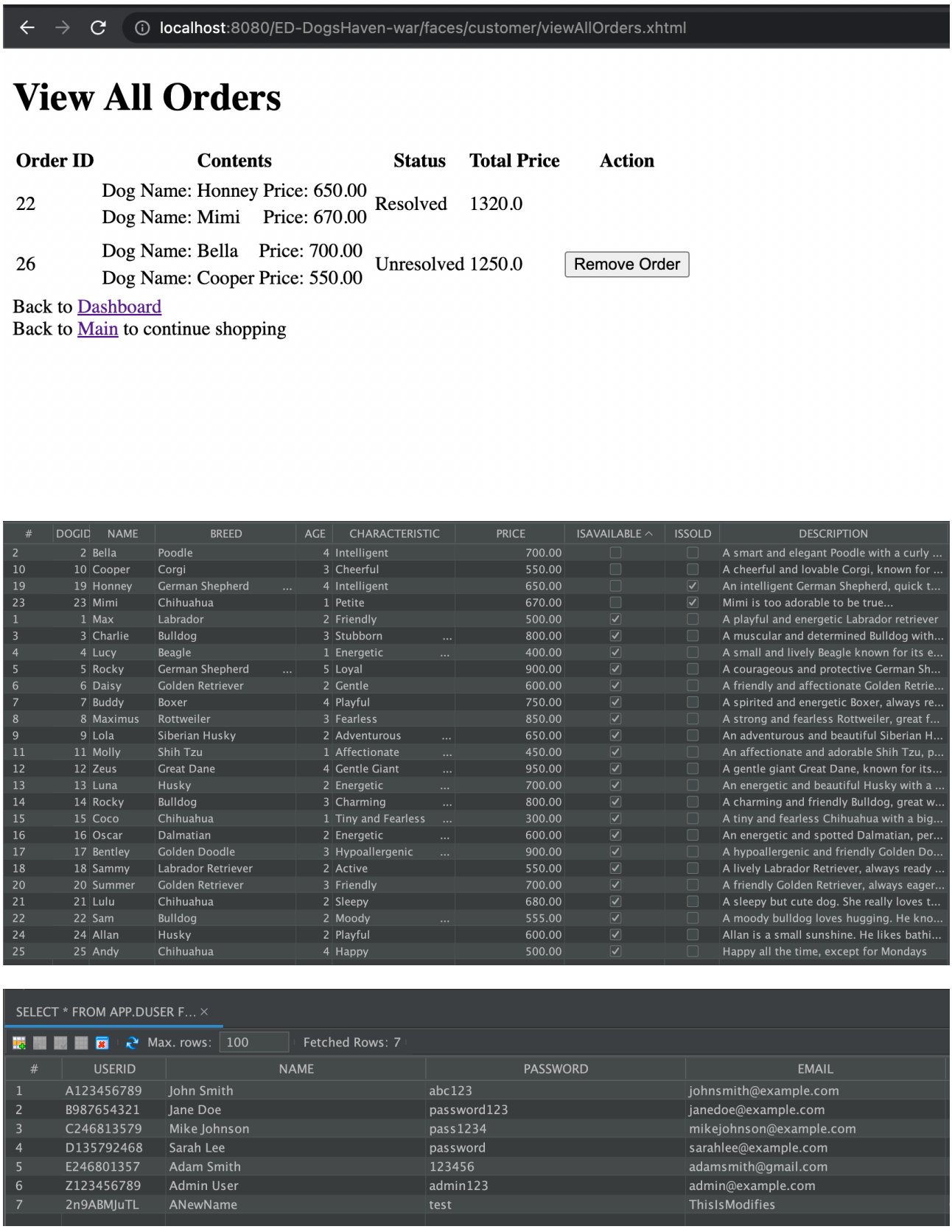
Back to Main to continue shopping



Checkout Successful

Your order has been successfully submitted.

Back To Dashboard


localhost:8080/ED-DogsHaven-war/faces/customer/viewAllOrders.xhtml
View All Orders
Order ID
Contents
Status
Total Price
Action
22
Dog Name: Honney Price: 650.00
Dog Name: Mimi Price: 670.00
Resolved
1320.0
26
Dog Name: Bella Price: 700.00
Dog Name: Cooper Price: 550.00
Unresolved
1250.0
Remove Order
Back to Dashboard
Back to Main to continue shopping
DOGID NAME BREED AGE CHARACTERISTIC PRICE ISAVAILABLE ISSOLD DESCRIPTION
2 2 Bella Poodle 4 Intelligent 700.00 A smart and elegant Poodle with a curly ...
10 10 Cooper Corgi 3 Cheerful 550.00 A cheerful and lovable Corgi, known for ...
19 19 Honney German Shepherd ... 4 Intelligent 650.00 An intelligent German Shepherd, quick t...
23 23 Mimi Chihuahua 1 Petite 670.00 Mimi is too adorable to be true...
1 1 Max Labrador 2 Friendly 500.00 A playful and energetic Labrador retriever
3 3 Charlie Bulldog 3 Stubborn ... 800.00 A muscular and determined Bulldog with...
4 4 Lucy Beagle 1 Energetic ... 400.00 A small and lively Beagle known for its e...
5 5 Rocky German Shepherd ... 5 Loyal 900.00 A courageous and protective German Sh...
6 6 Daisy Golden Retriever 2 Gentle 600.00 A friendly and affectionate Golden Retrie...
7 7 Buddy Boxer 4 Playful 750.00 A spirited and energetic Boxer, always re...
8 8 Maximus Rottweiler 3 Fearless 850.00 A strong and fearless Rottweiler, great f...
9 9 Lola Siberian Husky 2 Adventurous ... 650.00 An adventurous and beautiful Siberian H...
11 11 Molly Shih Tzu 1 Affectionate ... 450.00 An affectionate and adorable Shih Tzu, p...
12 12 Zeus Great Dane 4 Gentle Giant ... 950.00 A gentle giant Great Dane, known for its...
13 13 Luna Husky 2 Energetic ... 700.00 An energetic and beautiful Husky with a ...
14 14 Rocky Bulldog 3 Charming ... 800.00 A charming and friendly Bulldog, great w...
15 15 Coco Chihuahua 1 Tiny and Fearless ... 300.00 A tiny and fearless Chihuahua with a big...
16 16 Oscar Dalmatian 2 Energetic ... 600.00 An energetic and spotted Dalmatian, per...
17 17 Bentley Golden Doodle 3 Hypoallergenic ... 900.00 A hypoallergenic and friendly Golden Do...
18 18 Sammy Labrador Retriever 2 Active 550.00 A lively Labrador Retriever, always ready ...
20 20 Summer Golden Retriever 3 Friendly 700.00 A friendly Golden Retriever, always eager...
21 21 Lulu Chihuahua 2 Sleepy 680.00 A sleepy but cute dog. She really loves t...
22 22 Sam Bulldog 2 Moody ... 555.00 A moody bulldog loves hugging. He kno...
24 24 Allan Husky 2 Playful 600.00 Allan is a small sunshine. He likes bathi...
25 25 Andy Chihuahua 4 Happy 500.00 Happy all the time, except for Mondays
SELECT * FROM APP.DUSER F...
Max. rows: 100
Fetched Rows: 7
USERID NAME PASSWORD EMAIL
1 A123456789 John Smith abc123 johnsmith@example.com
2 B987654321 Jane Doe password123 janedoe@example.com
3 C246813579 Mike Johnson pass1234 mikejohnson@example.com
4 D135792468 Sarah Lee password sarahlee@example.com
5 E246801357 Adam Smith 123456 adamsmith@gmail.com
6 Z123456789 Admin User admin123 admin@example.com
7 2n9ABMJuTL ANewName test ThisIsModifies

SELECT * FROM APP.DORDER ... ×

Max. rows: 100 Fetch

#	ORDERID	CUSTOMERID
1	22	2n9ABMJuTL
2	26	2n9ABMJuTL

Max. rows: 100 Fetche

#	ORDERID	DOGID
1	22	19
2	22	23
3	26	2
4	26	10