FIT5192 Assignment One - Design Report

Car Dealer

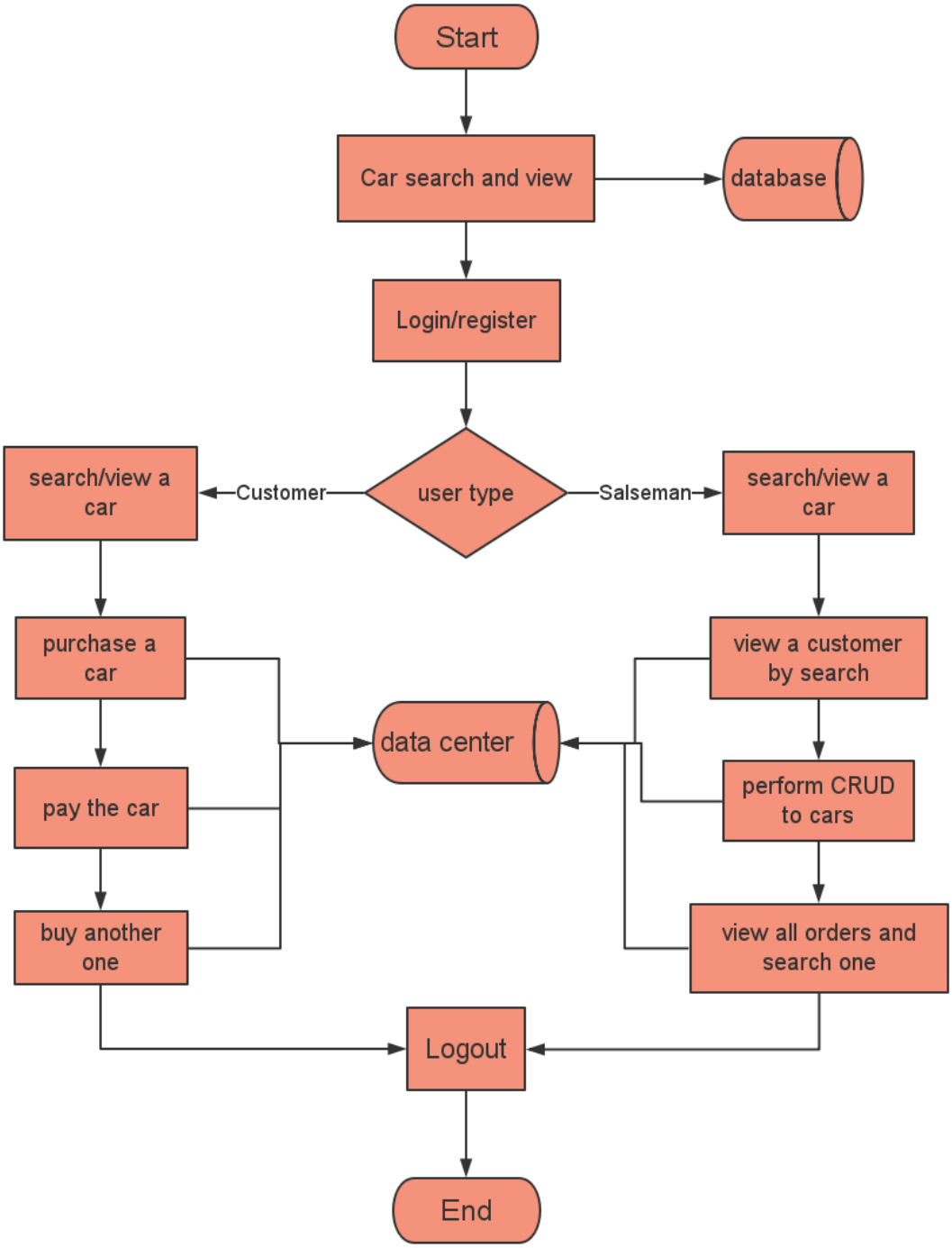
SHIJIN ZHAO: 27315894

Contents **Page**

1. **Overview**

   This application can do benefits to both customers and salesman. Before login, users can visit it and do a car search, then view the details of a car. If you want to buy a car you have to login and you can register an account in care you do not have one. When you login as a customer, you can buy a car then enter into your personal center to pay it. Another case, you login as a salesman, you can use this application to search customers to check their information including their sale orders. Salesman can also do CRUD operation to cars and view the whole orders and search one if you wish. Both users can logout by a command button.

2. **Functional diagram**


Start
Car search and view
database
Login/register
user type
Customer
Salseman
search/view a car
search/view a car
purchase a car
pay the car
buy another one
data center
view a customer by search
perform CRUD to cars
view all orders and search one
Logout
End

3. **Core program functionality** (description of core program functionality and how it works.)

3.1 let's look at the Car entity first.

```
@Override

public List<Car> queryCar(String manufacturer, String modelName, String modelNo, int carType) {

        Query query = em.createQuery("select c from Car c where c.modelName = :modelName AND c.modelNo = :modelNo AND c.manufacturer = :manufacturer AND c.carType = :carType");

        query.setParameter("modelName", modelName);

        query.setParameter("modelNo", modelNo);

        query.setParameter("manufacturer", manufacturer);

        query.setParameter("carType", carType);

        return query.getResultList();

    }
```

This is from JPACarRepository.

```
void  displayAllCars();


    void clearComboBoxes();


    void dispalyMessageInDialog(String message);


    void displayCarDetails(Car car);


    void displaySelectedCarDetails(Car car);
```

```
void displayCarDetails(List<Car> cars);

void displayCarDetails(Set<Car> cars);

String getSelectedCarId() throws Exception;

JButton getCloseButton();

JButton getSearchButton();

JButton getViewButton();

public JTable getCarTable();

Car getCarDetails();
```

This is pieces from my GUI interface class.

```
private void displayAllCars() {
        try {
            List<Car> cars;
            cars = carRepository.queryAll();
            if (cars != null) {
                this.gui.displayCarDetails(cars);
```

```
            }


        } catch (Exception ex) {
            this.gui.dispalyMessageInDialog("Failed to retrive cars: " + ex.getMessage());
        }
    }


    private void searchCar() {
        Car car = this.gui.getCar();



        List<Car> cars = carRepository.queryCar(car.getManufacturer(),
car.getModelName(), car.getModelNo(), car.getCarType());
        if(!cars.isEmpty()){
            gui.displayCarDetails(cars);
        }
    }
```

This pieces are from drive class.

Let' s look at codes from Web pages.

Here are codes about search.

```
@Override
    public Users queryUser(String email, String password) {
//        System.out.println("hellllll");
```

```
        Query query = entityManager.createQuery("select u from Users u where
u.emailString = :email and u.password = :password");

        query.setParameter("email", email);

        query.setParameter("password", password);

        List<Users> users = query.getResultList();

        if (users.isEmpty()) {

            return null;

        } else {

            return (Users) query.getResultList().get(0);

        }


    }



  @Override

    public List<Sales> salesmanQuery(int id, String carID) {

        Query query = entityManager.createQuery("SELECT s FROM Sales s WHERE
s.id = :id AND s.CarId = :CarId");

        query.setParameter("id", id);

        query.setParameter("CarId", carID);

        return query.getResultList();

    }
```

Let ‘s look at some codes from managed beans.

```
  public String paymoney(){
```

```
        for (Sales sale1 : salesByCustomer) {
            if (sale1.isOrderState()) {
                if (sale1 != null) {
                    sale1.setOrderState(false);
                    saleRepository.updateOrder(sale1);
                    return messageString = "payment succeed!";
                }
            }
        }
        return messageString = "update failed!";
    }

public String getOrderedCarId(){
    String orderCarId = FacesContext.getCurrentInstance()
            .getExternalContext().getRequestParameterMap()
            .get("CarId");
    return orderCarId;
    }

    public String getCurrentTime(){
        SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        return df.format(new Date());// new Date()
    }
```

Car management bean.

```
public String removeCar(){
        VIN =car.getVIN();
        if (null != this.VIN) {
            carRepository.removeCar(this.VIN);
            return messageString = "delete successfully!";
        }else{
          return messageString = "delete failed!";
        }
    }

    public  String updateCar(){
        if (this.car != null) {
            carRepository.updateCar(this.car);
            return messageString = "update successfully!";
        }else{
          return messageString = "update failed!";
        }

    }

public List<Car> searchCar(){
        cars = carRepository.queryCar(manufacture, modelName, modelNo, type);
        if (!cars.isEmpty()) {
            return cars;
```

```
        }else
            return null;
    }

    public String searchCarBySalesman(){
        car = carRepository.QueryCar(VIN);
        if (null != car) {
            return "car-management";
        }else{
            return messageString = "Can not find the car!";
        }
    }

    public void createCar(){
        Car newcar = new Car(VIN, type, thumnail, description, previewURL, carState,
modelNo, modelName, manufacture);
        carRepository.createCar(newcar);
    }
```

User managed bean.

```
public String logOut() {
        String outCome = "login";
        FacesContext fc = FacesContext.getCurrentInstance();
        HttpSession Session = (HttpSession) fc.getExternalContext()
                .getSession(false);
```

```
        Session.invalidate();
        return outCome;
    }
    public String login(){
        user = userRepository.queryUser(emailString, password);
        if (null != user) {
            if (user.getUserType()==1) {
                return "salesman-info";
            }
            return "customer-info";
        }else
            return messageString = "Login failed, please check your name or password!";

    }
    public String customerQuery(){
        Users result = userRepository.queryUserBySalesman(id, lastName, firstName,
userType, emailString);
        if (null != result) {
            return "customer-info";

        }else{
            return messageString = "The customer does not exit!";
        }
    }
```

Lastly, some pieces about JSF pages.

```
<div class="wrap">
    <h:body>
        <header id="header" class="clearfix">
        <a href="index.xhtml" id="logo"> <img src="images/logo.png" alt="car dealer"/> </a>
        <h:form>
            <nav id="navigation" class="navigation">
            <ul>
                <li><a href="index.xhtml">Home </a></li>
                <li><a href ="all-cars.xhtml">All Cars</a></li>
<!--                <li><h:commandLink id="continue"
                            action="all-cars"
                            immediate="true">
                        <a>All Car</a>
                    </h:commandLink></li>-->
                <li><a href="contact.xhtml">Contact</a></li>
                <li><a href="register.xhtml">Register </a></li>
                <li class="current-menu-item"><a href="login.xhtml">Login </a></li>
                <li><a> </a></li>
            </ul>
        </nav>
        </h:form>
```

```
        </header>
      <div class="top-panel clearfix">
        <div class="widget_custom_register">
          <h3 class="widget-title">User Login</h3>
          <h:form  id="boxpanel" class="form-panel">
            <fieldset>
              <label for="email">Email-address</label>
              <h:inputText id="email" label="emailAddress"
                           value="#{registerManagedBean.emailString}"
                           required="true"
                           requiredMessage="You must enter your
EmailAddress."></h:inputText>
            </fieldset>
            <fieldset>
              <label for="password">Password</label>
              <h:inputSecret id="password" label="password"
                            value="#{registerManagedBean.password}"
                            required="true"
                            requiredMessage="">
              </h:inputSecret>
            </fieldset>
            <div class="clear"></div>
            <h:commandButton id="submitLogin" class="submit-search"
```

```
                    value="Login" action="#{registerManagedBean.login}">

                </h:commandButton>

                <p> <h:outputText class="outputError"
value="#{registerManagedBean.messageString}"/>

                </p>



              </h:form>

            </div> <!-- .formPanel-->

                    </div>

                <footer id="footer" class="container">

              <p class="copyright">Copyright &copy; 2016. By Alex. All rights
reserved</p>

            </footer>

          </h:body>

    </div>
```

Sorry for the format because something is wrong with my sublime. I can not copy codes as html.

4. **Usability Design Review**

Users can login and logout and register.

UI of web pages is good I think.

5. Checklist of site functionality

1. Pass Functionality	
Search for Car by	√
Make	√
Model,	√
Model No	√
Type	√
Results with tabular format with heading.	√
Option to view the full details	√
Additional Pass Requirement	
Learning Summary Report	√
2. Credit Functionality	
Login using a username and password	√
Sales people can: View	√
Add	√
Update	√
Delete cars	√
3. Distinction Functionality	
Customers can buy car(s)	√
Sales people search Customers by a **combination** of ID, last name, first name, type and email.	√
When adding a car to the system, the information of the car can be obtained via web service	
4. High Distinction Requirements	
This Design Document	
6. Technical Requirements	
Pass	
JSF web clients	√
GUI Swing application clients	√
Persistence API	√
Application managed entity manager or container managed entity manager.	√
Credit	
ONLY web client is required	
Interaction between clients and database handled by EJBs	√
BOTH Criteria API and JPQL	√
Distinction	
Ability of mapping inheritance to database must be demonstrated.	
Bean validations used to validate data.	√
Consumption of web services conducted in EJBs.	
Application secured using JAAS API.	
Audit	
No breaking of copyright	√

Optional requirements (for Highest range of Distinction, once above requirements satisfied)

6. **User stories (that are driving your design decisions)**

7. **Data dictionary** (of your application, including the main data structures and types used in your application.)