

Abstract

In the digital age , every bit of information is available to us on the internet ,ranging from entertainment timings like movie timings ,sports timings to work like flight and train schedules .This is a humble attempt a building a flight schedule website using Spring MVC. HTML was also used for front end development.

AIRLINE TRAVEL website WITH SPRING MVC

A web based flight schedule displaying website built using Spring

Amartya Choudhury

JU-BCSE IV

Roll – 001610501026

Internet Technology Lab

amartyachowdhury98@gmail.com

Problem Statement:

Implement a web application for “Travel Thru Air” based on Spring MVC framework to support any of the following two use cases

1. A list of current special deals must appear on the home page. Each special deal must display the departure city, the arrival city, and the cost. These special deals are set up by the marketing

department and change during the day, so it can’t be static. Special deals are only good for a limited amount of time.

2. A user may search for flights, given a departure city, time and an arrival city. The results must display the departure city, the arrival city, the total cost, and how many legs the flight will have.

Provide a login controller for user login facility. The solution should reflect each layer of the spring framework.

Generate trace using log4j.

Key Strategies of Spring :

1. Spring is a light weight framework and It minimally invasive development with POJO.
2. Spring achieves the loose coupling through dependency injection and [interface based programming](https://en.wikipedia.org/wiki/Interface-based_programming).
3. Spring supports declarative programming through aspects and common conventions.
4. [Boilerplate reduction](https://en.wikipedia.org/wiki/Boilerplate_code) through aspects and templates.

Above are the 4 key strategies of spring framework simplifying the development of Java enterprise application.

Advantages of Spring Framework:

The Spring framework addresses most of the infrastructure functionalities of the Enterprise applications. Following are the few major advantages of Spring Framework.

1. Spring enables the developers to develop enterprise applications using POJOs (Plain Old Java Object). The benefit of developing the applications using POJO is, that we do not need to have an enterprise container such as an application server but we have the option of using a robust servlet container.
2. Spring provides an abstraction layer on existing technologies like servlets, jsps, jdbc, jndi, rmi, jms and Java mail etc., to simplify the develpment process.
3. Spring comes with some of the existing technologies like ORM framework, logging framework, J2EE and JDK Timers etc, Hence we don’t need to integrate explicitly those technologies.
4. Spring WEB framework has a well-designed  web MVC framework, which provides a great alternate to lagacy web framework.
5. Spring can eliminate the creation of the singleton and factory classes.
6. Spring provides a consistent transaction management interface that can scale down to a local transaction and scale up to global transactions (using JTA).
7. Spring framework includes support for managing business objects and exposing their services to the presentation tier components, so that the web and desktop applications can access the same objects.
8. Spring framework has taken the best practice that have been proven over the years in several applications and formalized as design patterns.
9. Spring application can be used for the development of different kind of applications, like standalone applications, standalone GUI applications, Web applications and applets as well.
10. Spring supports both xml and anotation configurations.
11. Spring Framework allows to develop standalone, desktop, 2 tire – n-tire architecture and distributed applications.
12. Spring gives built in middleware services like Connection pooling, Transaction management and etc.,
13. Spring provides a light weight container which can be activated without using webserver or application server.



Fig 1 : Spring Web MVC workflow

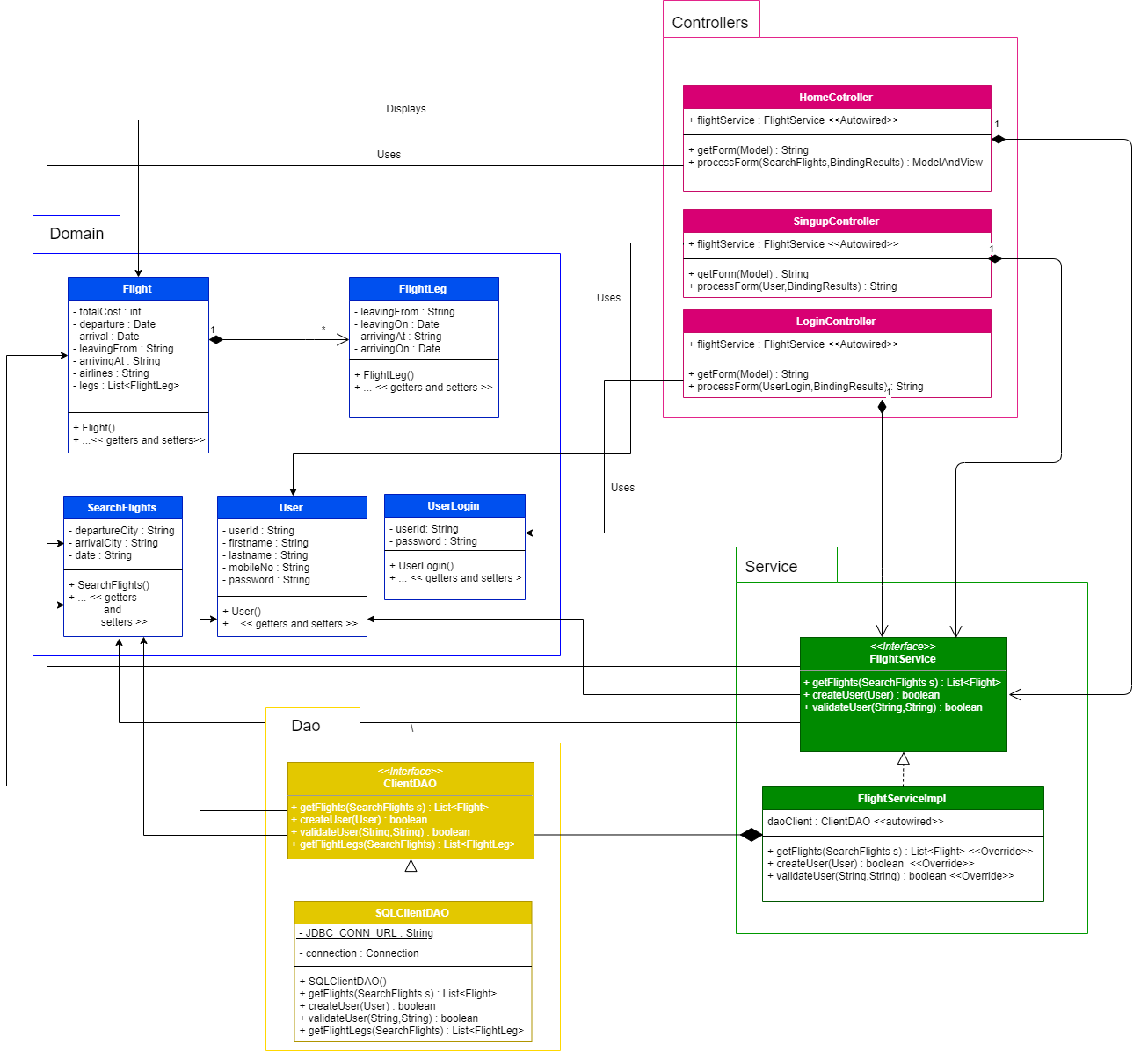
Design:

Fig 2: Class Diagram

Configuration Files:

The front-controller *DispatcherServlet* is configured with servlet name “spring” . All requests(“/”) are forwarded to *DispatcherServlet* “spring” . The web specific beans are declared in “spring-servlet.xml”. For mapping view-names to views , path prefix and path suffix are declared .The packages are scanned using <contex:component-scan base-package=”” > tag.

web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  id="WebApp\_ID" version="3.1">  <display-name>XXXX</display-name>  <absolute-ordering/>  <servlet>  <servlet-name>spring</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>spring</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>  /WEB-INF/spring-servlet.xml  </param-value>  </context-param>  <listener>  <listener-class>  org.springframework.web.context.ContextLoaderListener  </listener-class>  </listener>    </web-app> |

spring-servlet.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd*  *http://www.springframework.org/schema/mvc*  *http://www.springframework.org/schema/mvc/spring-mvc.xsd"*>    <context:component-scan base-package=*"controllers"* />  <context:component-scan base-package=*"dao"* />  <context:component-scan base-package=*"domain"* />  <context:component-scan base-package=*"service"* />  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"*>  <value>/WEB-INF/view/</value>  </property>  <property name=*"suffix"*>  <value>.jsp</value>  </property>  </bean>  <mvc:annotation-driven/> |

Views :

Views for login ,signup and home page are defined.

signup.jsp

|  |
| --- |
| **<%@** page language="java" contentType="text/html; charset=ISO-8859-1"  pageEncoding="ISO-8859-1"**%>**  **<%@** taglib prefix="form" uri="http://www.springframework.org/tags/form"**%>**  <!DOCTYPE html>  <html>  <head>  <meta charset="ISO-8859-1">  <title>Signup</title>  <style>  .error{color:red}  </style>  </head>  <body>  <h1>Signup</h1>  <form:form action="/TravelThruAir\_Spring/signup/processForm" modelAttribute="user">  <Label>UserId :</Label><br>  <form:input path ="userId" type = "text" /><form:errors path="userId" cssClass="error"/><br>    <Label>Firstname :</Label><br>  <form:input path ="firstname" type = "text" /><form:errors path="firstname" cssClass="error"/><br>  <Label>Lastname :</Label><br>  <form:input path ="lastname" type = "text" /><form:errors path="lastname" cssClass="error"/><br>  <Label>Mobile No :</Label><br>  <form:input path ="mobileNo" type = "text" /><form:errors path="mobileNo" cssClass="error"/><br>  <Label>Password :</Label><br>  <form:input path ="password" type = "password" /><form:errors path="password" cssClass="error"/><br>  <input type="submit" value="submit">  </form:form>  <a href="./login">Login</a>  </body>  </html> |

login.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <%@ taglib prefix=*"form"* uri=*"http://www.springframework.org/tags/form"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Login</title>  <style>  *.errors*{color:*red*}  </style>  </head>  <body>  <h1>Login</h1>  <form:form action=*"/TravelThruAir\_Spring/login/processForm"* modelAttribute=*"user"*>  <Label>UserId :</Label><br>  <form:input path =*"userId"* type = *"text"* /><form:errors path=*"userId"* cssClass=*"errors"*/><br>  <Label>Password :</Label><br>  <form:input path =*"password"* type = *"password"* /><form:errors path=*"password"* cssClass=*"errors"*/><br>  <input type=*"submit"* value=*"submit"*>  </form:form>    <a href=*"./signup"*>signup</a>  </body>  </html> |

welcome.jsp

|  |
| --- |
| <%@ taglib prefix=*"form"* uri=*"http://www.springframework.org/tags/form"*%>  <%@page import=*"java.util.List"* %>  <%@page import=*"domain.\*"* %>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Home</title>  <style>  *.errors*{color:*red*}  </style>  </head>  <body>  <h1>Welcome ${user.userId} to Travel Thru Air</h1>  <h3>Search for flights</h3>  <form:form action=*"/TravelThruAir\_Spring/home/processForm"* modelAttribute = *"search"*>  <Label>Departure :</Label><br>  <form:input path = *"departureCity"* type=*"text"*/><form:errors path=*"departureCity"* cssClass=*"errors"*/><br>  <Label>Arrival :</Label><br>  <form:input path = *"arrivalCity"* type=*"text"*/><form:errors path=*"arrivalCity"* cssClass=*"errors"*/><br>  <Label>Date :</Label><br>  <form:input path = *"date"* type=*"text"*/><form:errors path=*"date"* cssClass=*"errors"*/><br>  <input type=*"submit"* value=*"submit"*><br>  </form:form>  <h2>Flights</h2>  <table border=*"1"*>  <tr>  <th>Flight Number</th>  <th>Airline</th>  <th>Departing</th>  <th>Arrival</th>  <th>TicketPrice(Rs.)</th>  </tr>  <%  **if**(request.getAttribute("flights") != **null**){  List<Flight> flights= (List<Flight>)request.getAttribute("flights");  **for**(Flight f:flights){  out.print("<tr>");  out.print("<td>");  out.print(f.flightID);  out.print("</td>");  out.print("<td>");  out.print(f.airlines);  out.print("</td>");  out.print("<td>");  out.print(f.getDeparture());  out.print("</td>");  out.print("<td>");  out.print(f.getArrival());  out.print("</td>");  out.print("<td>");  out.print(f.getTotalCost());  out.print("</td>");  out.print("</tr>");  **if**(f.getLegs()!= **null** && f.getLegs().size() > 1){  List<FlightLeg> legs = f.getLegs();  out.print("<tr class=\"break\"><td colspan=\"5\">Legs</td></tr>");  out.print("<table border=\"1\"><tr><th>Leaving from</th><th>Leaving On</th><th>Arriving At</th><th>Arriving On</th>");  **for**(FlightLeg leg: legs){  out.print("<tr>");  out.print("<td>");  out.print(leg.getLeavingFrom());  out.print("</td>");  out.print("<td>");  out.print(leg.getLeavingOn());  out.print("</td>");  out.print("<td>");  out.print(leg.getArrivingAt());  out.print("</td>");  out.print("<td>");  out.print(leg.getArrivingOn());  out.print("</td>");  out.print("</tr>");  }  }  }  }  %>  </table>  <a href=*"/TravelThruAir\_Spring/login"*>Logout</a>  </body>  </html> |

invalidCredentials.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Invalid Credentials</title>  </head>  <body>  <h2>Sorry Invalid username or password</h2>  <a href=*"/TravelThruAir\_Spring/login"*>back</a>  </body>  </html> |

usernameExists.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"ISO-8859-1"*>  <title>Username exists</title>  </head>  <body>  <h2>Sorry! that username is taken</h2>  <a href=*"/TravelThruAir\_Spring/signup"*>back</a>  </body>  </html> |

Controllers

All controllers have a “flightService” property which is autowired . This provides loose coupling between the service and the web layer.

SignupController.jsp

|  |
| --- |
| package controllers;  import javax.validation.Valid;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.validation.BindingResult;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.servlet.ModelAndView;  import domain.\*;  import service.\*;  @Controller  @RequestMapping("/signup")  public class SignupController {  @Autowired  FlightService flightService;  @RequestMapping("")  public String getForm(Model model) {  User u = new User();  model.addAttribute("user",u);  return "signup";  }  @RequestMapping("/processForm")  public String processForm(@Valid @ModelAttribute("user") User u,BindingResult br) {  if(br.hasErrors()) return "signup";  if(flightService.createUser(u)) return "redirect:/home";  return "usernameExists";  }  } |

LoginController.java

|  |
| --- |
| package controllers;  import javax.validation.Valid;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.validation.BindingResult;  import org.springframework.web.bind.annotation.ModelAttribute;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.servlet.ModelAndView;  import domain.\*;  import service.\*;  @Controller  @RequestMapping("/login")  public class LoginController {  @Autowired  FlightService flightService;  @RequestMapping("")  public String getForm(Model model) {  UserLogin u = new UserLogin();  model.addAttribute("user",u);  return "login";  }  @RequestMapping("/processForm")  public String processForm(@Valid @ModelAttribute("user") UserLogin u,BindingResult br) {  System.out.println(u.getUserId().length());  if(br.hasErrors()) {  System.out.println(br.toString());  return "login";  }  if(flightService.validateUser(u.getUserId(),u.getPassword())) return "redirect:/home";  else return "invalidCredentials";  }  } |

HomeController.java

|  |
| --- |
| **@**Controller  @RequestMapping("/home")  public class HomeController {  FlightService flightservice;  @Autowired  public void setFlightService(FlightService fs) {  flightservice = fs;  }  @RequestMapping("")  public String getForm(Model model) {  SearchFlights s= new SearchFlights();  model.addAttribute("search",s);  System.out.println("in home getForm");  return "home";  }  @RequestMapping("/processForm")  public ModelAndView processForm(@Valid @ModelAttribute("search") SearchFlights s,BindingResult br) {  if(br.hasErrors()) {    ModelAndView mav = new ModelAndView("home");  return mav;  }  List<Flight> flights = flightservice.getFlights(s);  ModelAndView mav = new ModelAndView("home");  mav.addObject("flights",flights);  return mav;  }  } |

DAO:

The ClientDAO sets up the blueprint for the DAO layer. The SQLClientDAO, implementation connects to a relational MSSQL database. There are 2 tables used- i. Flight(flightId, start,departTime,end,arrivalTime,totalCost,airlines)

ii .FlightLeg(flightId(f.k),departFrom,departOn,arriveAt,arriveOn)

ClientDAO.java

|  |
| --- |
| package dao;  import domain.\*;  import java.util.List;  import org.springframework.stereotype.Component;  @Component  public interface ClientDAO {  List<Flight> getFlights(SearchFlights s);  List<FlightLeg> getFlightLegs(String flightId);  boolean createUser(User u);  boolean validateUser(String userId,String password);  } |

SQLClientDAO.java

|  |
| --- |
| package dao;  import java.util.List;  import org.springframework.stereotype.Component;  import java.util.ArrayList;  import domain.Flight;  import domain.SearchFlights;  import domain.FlightLeg;  import domain.User;  import java.sql.\*;  import java.text.ParseException;  import java.text.SimpleDateFormat;  @Component  public class SQLClientDAO implements ClientDAO {  private static final String JDBC\_CONNECTION\_URL = "jdbc:sqlserver://localhost\\MSSQLSERVER:60768;databaseName=travelThruAir;user=achoudhury98;password=1234";;  private static final String GET\_FLIGHTS = "SELECT \* FROM Flight WHERE departure = ? AND arrival = ? AND CAST(departureTime AS Date) = ?";  private static final String GET\_FLIGHT\_LEGS = "SELECT departFrom,departOn,arriveAt,arriveOn FROM FlightLeg WHERE flightID = ? ";  private static final String CREATE\_USER = "INSERT INTO Users Values(?,?,?,?,?)";  private static final String VALIDATE\_USER = "SELECT \* FROM Users WHERE userId = ? AND password = ?";  private Connection connection;  SQLClientDAO(){  try {  Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");  connection = DriverManager.getConnection(JDBC\_CONNECTION\_URL);  }  catch (SQLException e) {  e.printStackTrace();  }  catch(ClassNotFoundException e) {  e.printStackTrace();  }  }    @Override  public boolean createUser(User u) {  boolean complete = false;  try {  PreparedStatement p = connection.prepareStatement(CREATE\_USER);  p.setString(1, u.getId());  p.setString(2, u.firstname());  p.setString(3, u.lastname());  p.setString(4, u.mobileNo());  p.setString(5, u.password());  p.executeUpdate();  complete = true;  }  catch(SQLException e) {  e.printStackTrace();  }  return complete;  }    @Override  public boolean validateUser(String userId,String password) {  boolean validated = false;  try {  PreparedStatement p = connection.prepareStatement(VALIDATE\_USER);  p.setString(1,userId);  p.setString(2, password);  ResultSet res = p.executeQuery();  if(res.next()) validated = true;  }  catch(SQLException e) {    }  return validated;  }  @Override  public List<Flight> getFlights(SearchFlights s){  System.out.println(s.getDepartureCity() + s.getArrivalCity() +s.getDate().toString());  List<Flight> flights = null;  try {  PreparedStatement p = connection.prepareStatement(GET\_FLIGHTS);  p.setString(1, s.getDepartureCity());  p.setString(2, s.getArrivalCity());  SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");    p.setDate(3,new java.sql.Date(format.parse(s.getDate()).getTime()));  ResultSet res = p.executeQuery();  while(res.next()) {  if(flights == null) flights = new ArrayList<Flight>();  flights.add(new Flight(res.getString(1),res.getString(2),res.getString(3),new java.util.Date(res.getTimestamp(4).getTime()),new java.util.Date(res.getTimestamp(5).getTime()),res.getInt(6),res.getString(7)));  }  }  catch(SQLException e) {  e.printStackTrace();  } catch (ParseException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return flights;  }  @Override  public List<FlightLeg> getFlightLegs(String flightID){  List<FlightLeg> legs = null;  try {  PreparedStatement p = connection.prepareStatement(GET\_FLIGHT\_LEGS);  p.setString(1, flightID);  ResultSet res = p.executeQuery();  while(res.next()) {  if(legs == null) legs = new ArrayList<>();  legs.add(new FlightLeg(res.getString(1),new java.util.Date(res.getTimestamp(2).getTime()),res.getString(3),new java.util.Date(res.getTimestamp(4).getTime())));  }  }  catch(SQLException e) {  e.printStackTrace();  }  return legs;  }  } |

Service

FlightService method defines all the coarse grained functionality of the backend . It is structured in such a way that all method calls are independent (ie. No state preservation between calls).

FlightService.java

|  |
| --- |
| package service;  import java.util.List;  import org.springframework.stereotype.Component;  import domain.\*;  @Component  public interface FlightService {  List<Flight> getFlights(SearchFlights s);  boolean createUser(User u);  boolean validateUser(String userId,String password);  } |

The implementation of the flight service .It uses an instance of ClientDAO , which is again autowired. This provides loose coupling between the dao layer and the service .

FlightServiceImpl.java

|  |
| --- |
| package service;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Component;  import domain.\*;  import dao.\*;  @Component  public class FlightServiceImpl implements FlightService {  ClientDAO daoclient;    @Autowired  void setClient(ClientDAO client) {  this.daoclient = client;  }  @Override  public List<Flight> getFlights(SearchFlights s) {  List<Flight> flights = daoclient.getFlights(s);  try {  for(Flight flight:flights) {  flight.legs = daoclient.getFlightLegs(flight.flightID);  }  }  catch(NullPointerException e) {  e.printStackTrace();  }  return flights;  }  @Override  public boolean createUser(User u) {  return daoclient.createUser(u);  }  @Override  public boolean validateUser(String userId,String password) {  return daoclient.validateUser(userId, password);  }  } |

Domain:

A Flight class stands for a single flight instance . A flight has one or more flight legs .A SearchFlight is used to find flights by date , departure city and arrival city.A UserLogin is a class used for login , where user sends only userId and password

Flight.java

|  |
| --- |
| **package** domain;  **import** **java.util.List**;  **import** **java.util.ArrayList**;  **import** **java.util.Date**;  **import** **dao.\***;  **public** **class** **Flight** {  **public** String flightID;  **public** List<FlightLeg> legs;    **private** **int** totalCost;  **private** Date departure;  **private** Date arrival;  **private** String leavingFrom;  **private** String arrivingAt;  **public** String airlines;    **public** **Flight**(String flightID,String leavingFrom,String arrivingAt,Date departure,Date arrival,**int** totalCost,String airlines){  **this**.flightID = flightID;  **this**.leavingFrom = leavingFrom;  **this**.arrivingAt = arrivingAt;  **this**.departure = departure;  **this**.arrival = arrival;  **this**.totalCost = totalCost;  **this**.airlines = airlines;  legs = **null**;  }  **public** String **getFlightID**() {  **return** flightID;  }  **public** **void** **setFlightID**(String flightID) {  **this**.flightID = flightID;  }  **public** List<FlightLeg> **getLegs**() {  **return** legs;  }  **public** **void** **setLegs**(List<FlightLeg> legs) {  **this**.legs = legs;  }  **public** **int** **getTotalCost**() {  **return** totalCost;  }  **public** **void** **setTotalCost**(**int** totalCost) {  **this**.totalCost = totalCost;  }  **public** Date **getDeparture**() {  **return** departure;  }  **public** **void** **setDeparture**(Date departure) {  **this**.departure = departure;  }  **public** Date **getArrival**() {  **return** arrival;  }  **public** **void** **setArrival**(Date arrival) {  **this**.arrival = arrival;  }  **public** String **getLeavingFrom**() {  **return** leavingFrom;  }  **public** **void** **setLeavingFrom**(String leavingFrom) {  **this**.leavingFrom = leavingFrom;  }  **public** String **getArrivingAt**() {  **return** arrivingAt;  }  **public** **void** **setArrivingAt**(String arrivingAt) {  **this**.arrivingAt = arrivingAt;  }  **public** String **getAirlines**() {  **return** airlines;  }  **public** **void** **setAirlines**(String airlines) {  **this**.airlines = airlines;  }  **public** String **getFlightId**() {  **return** flightID;  }  } |

FlightLeg.java

|  |
| --- |
| package domain;  import java.util.Date;  public class FlightLeg {  String leavingFrom;  Date leavingOn;  String arrivingAt;  Date arrivingOn;  public FlightLeg(String leavingFrom,Date leavingOn,String arrivingAt,Date arrivingOn) {  this.leavingFrom = leavingFrom;  this.leavingOn = leavingOn;  this.arrivingAt = arrivingAt;  this.arrivingOn = arrivingOn;  }  public String getLeavingFrom() {  return leavingFrom;  }  public void setLeavingFrom(String leavingFrom) {  this.leavingFrom = leavingFrom;  }  public Date getLeavingOn() {  return leavingOn;  }  public void setLeavingOn(Date leavingOn) {  this.leavingOn = leavingOn;  }  public String getArrivingAt() {  return arrivingAt;  }  public void setArrivingAt(String arrivingAt) {  this.arrivingAt = arrivingAt;  }  public Date getArrivingOn() {  return arrivingOn;  }  public void setArrivingOn(Date arrivingOn) {  this.arrivingOn = arrivingOn;  }  } |

SearchFlight.java

|  |
| --- |
| package domain;  import java.util.Date;  import javax.validation.constraints.NotEmpty;  import javax.validation.constraints.NotNull;  import org.springframework.format.annotation.DateTimeFormat;  import java.text.ParseException;  import java.text.SimpleDateFormat;  public class SearchFlights {  @NotEmpty(message = "is required")  String departureCity;  @NotEmpty(message ="is required")  String arrivalCity;  @NotEmpty(message ="is required")  String date;  SearchFlights(String departure,String arrival,String dt){  SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy");  this.departureCity = departure;  this.arrivalCity = arrival;  try {  Date dat = formatter.parse(dt);  this.date=dt;  }  catch(ParseException e) {  e.printStackTrace();  }  }  public SearchFlights() {    }  public void setDepartureCity(String departureCity) {  this.departureCity = departureCity;  }  public void setArrivalCity(String arrivalCity) {  this.arrivalCity = arrivalCity;  }  public void setDate(String date) {  System.out.println(date);  SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");  this.date = date;  }  /\*public void setDate(Date date) {  this.date = date;  }\*/  public String getDepartureCity() {  return departureCity;  }  public String getArrivalCity() {  return arrivalCity;  }  public String getDate() {  return date;  }  } |

User.java

|  |
| --- |
| **package** domain;  **import** **javax.validation.constraints.NotEmpty**;  **import** **javax.validation.constraints.NotNull**;  **import** **javax.validation.constraints.Size**;  **public** **class** **User** {  **@NotEmpty**(message="is required")  String userId;  **@NotEmpty**(message="is required")  String firstname;  **@NotEmpty**(message="is required")  String lastname;  **@NotEmpty**(message="is required")  **@Size**(min=**10**, max=**10**,message="must be 10 digits")  String mobileNo;  **@NotEmpty**(message="is required")  String password;  **public** **User**(String userId,String firstname,String lastname,String mobileNo,String password){  **this**.userId = userId;  **this**.firstname = firstname;  **this**.lastname = lastname;  **this**.mobileNo = mobileNo;  **this**.password = password;  }  **public** **User**(){    }  **public** **User**(String userId,String password) {  **this**.userId = userId;  **this**.password = password;  }  **public** String **getId**() {  **return** userId;  }  **public** String **firstname**() {  **return** firstname;  }  **public** String **lastname**() {  **return** lastname;  }  **public** String **getUserId**() {  **return** userId;  }  **public** **void** **setUserId**(String userId) {  **this**.userId = userId;  }  **public** String **getFirstname**() {  **return** firstname;  }  **public** **void** **setFirstname**(String firstname) {  **this**.firstname = firstname;  }  **public** String **getLastname**() {  **return** lastname;  }  **public** **void** **setLastname**(String lastname) {  **this**.lastname = lastname;  }  **public** String **getMobileNo**() {  **return** mobileNo;  }  **public** **void** **setMobileNo**(String mobileNo) {  **this**.mobileNo = mobileNo;  }  **public** String **getPassword**() {  **return** password;  }  **public** **void** **setPassword**(String password) {  **this**.password = password;  }  **public** String **mobileNo**() {  **return** mobileNo;  }  **public** String **password**() {  **return** password;  }  } |

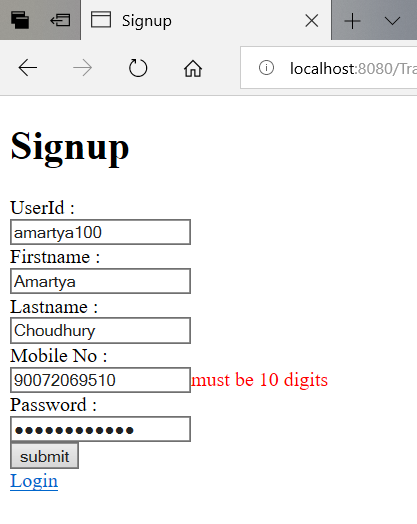
UserLogin.java

|  |
| --- |
| package domain;  import javax.validation.constraints.NotBlank;  import javax.validation.constraints.NotEmpty;  import javax.validation.constraints.NotNull;  public class UserLogin {  //@NotNull(message="is required")  @NotEmpty(message="User name cannot be empty")  String userId;  //@NotNull(message="is required")  @NotEmpty(message="Password cannot be empty")  String password;  public UserLogin(){  }  public String getUserId() {  return userId;  }  public void setUserId(String userId) {  this.userId = userId;  }  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }    } |

Form Validation:

The input provided by the user is validated before further processing is done .Hibernate validator is used for this purpose and the validation rule is hardcoded along with the property for a class . For eg the “mobileNo: property of a User must be of 10 letters and so an annotation @size(max=10 , min=10) is coded above it .Also <form:errors> tag in the jsp file show text that is to be displayed if error occurs . Spring provides automatic binding of form input to bean properties , but if an error occurs during validation , then the BindingResult object br would have errors and the same page is displayed , along with the error that occurred .

Output:

  Fig 2 : Signup form displaying validation

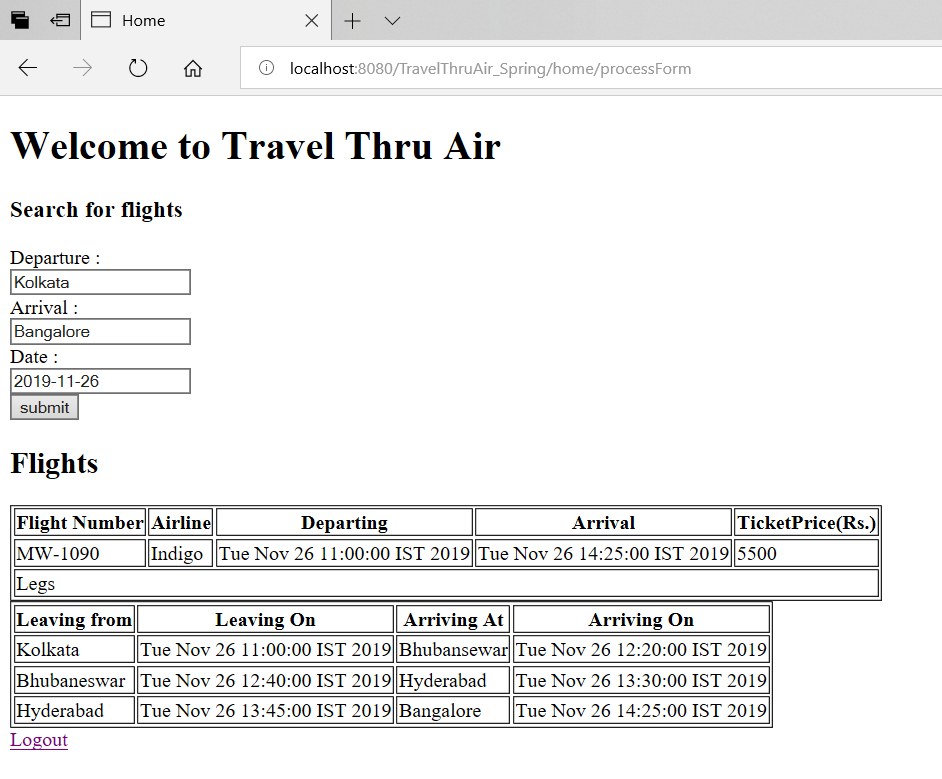


Fig 3 : Flight schedule showing breaks

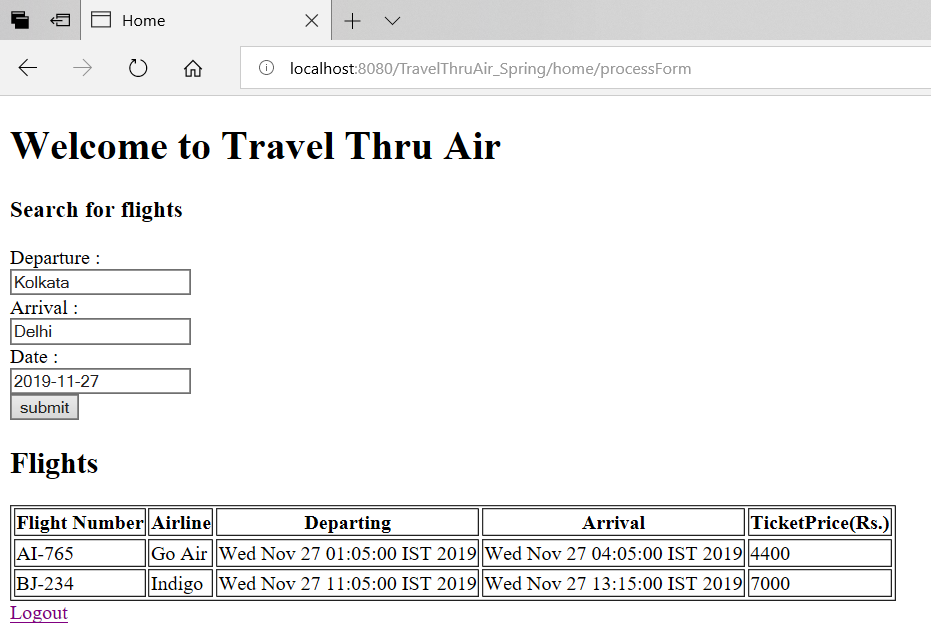
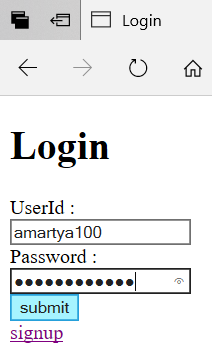


Fig 4: More flights without breaks

 Fig 5 : Login Form

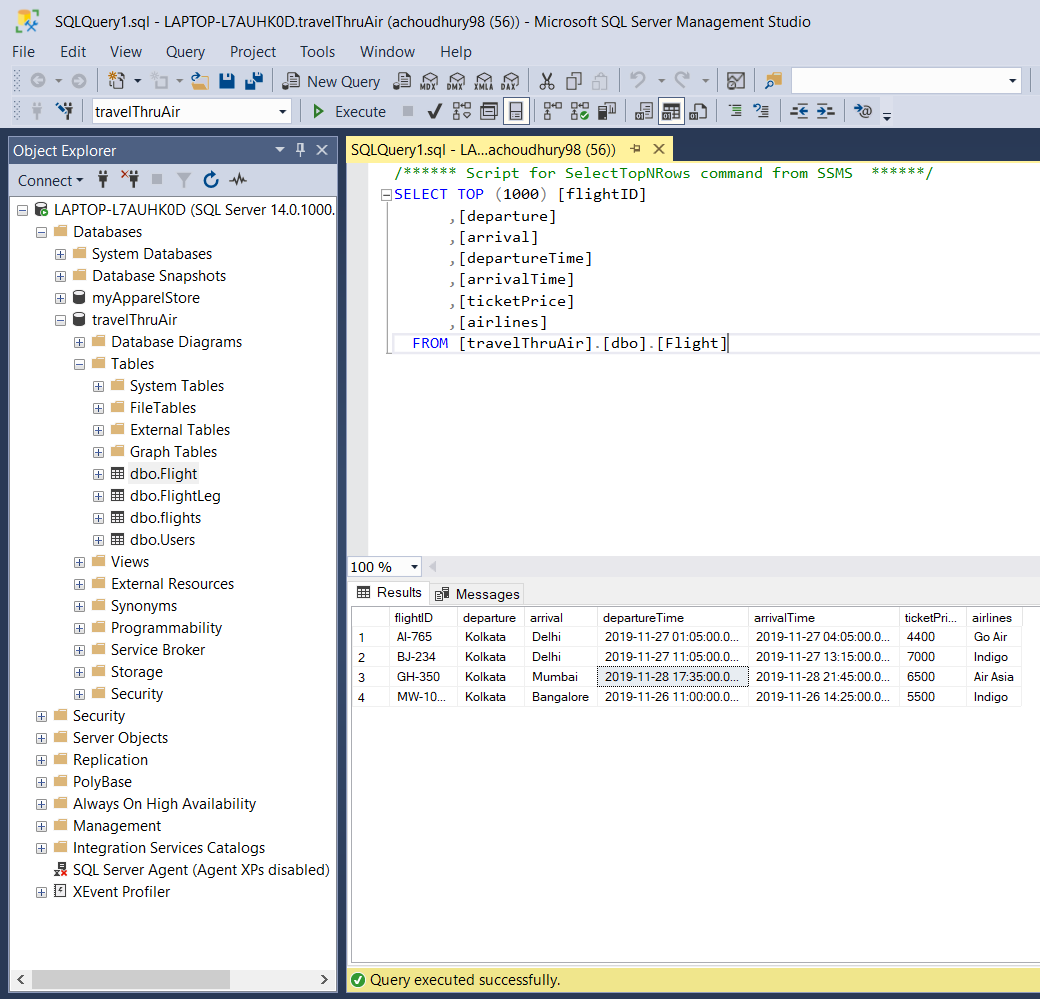
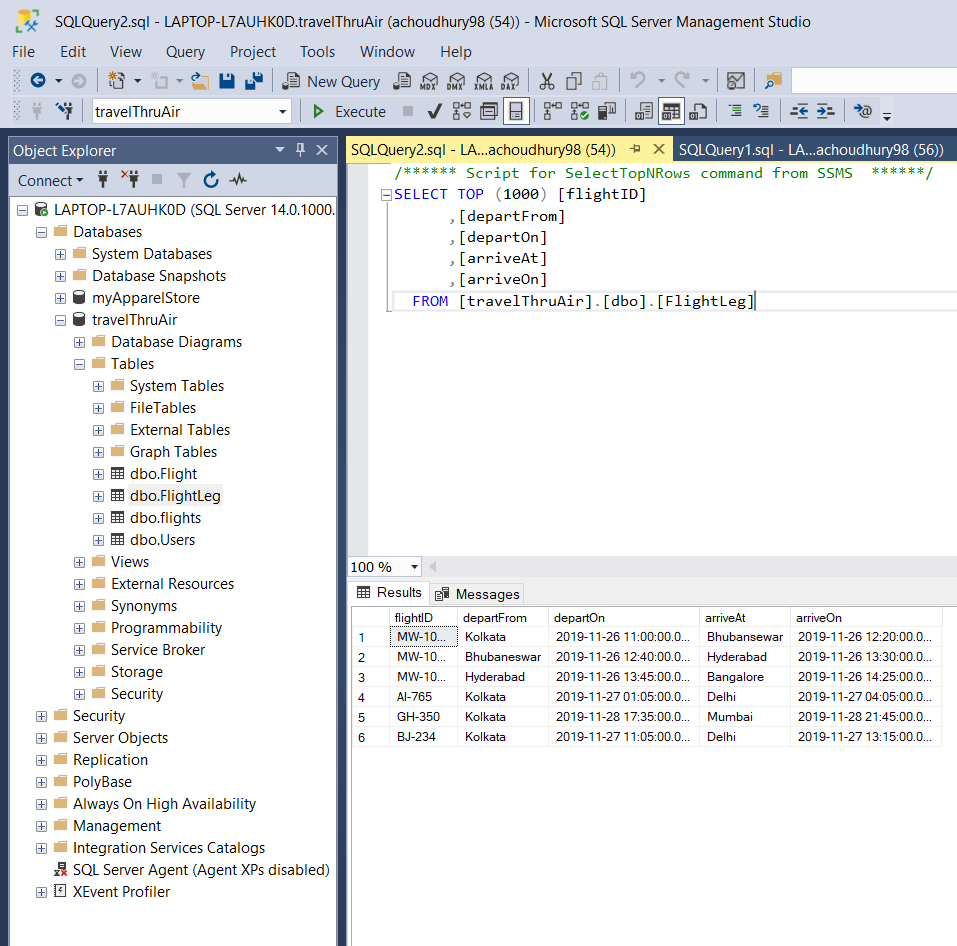
 Fig 6 : Flight table

Fig 7 : FlightLeg table 

References:

1. Expert Spring MVC and Web Flow - A