The Internet- and Web-enabled applications have revolutionized the way businesses are carried out. Remember the days when to buy tickets for a vacation, you had to call up ticket agents, find the airlines, search for deals, and so on? Well, since the Internet took off, things haven't been the same. Now you have many Web sites from both third-party vendors and airlines offering anything from cars to cruises to vacation packages, with deal comparisons and what have you. Clearly, the consumer is the winner!

As a part of your learning experience using WebLogic Server, you will build a simple airline ticket booking system, which will model a real-world, Web-enabled airline ticket booking system. You'll begin by briefly outlining the features that will be provided by your airline ticket booking system.

Application Features

Users of the airline ticket booking system will have the ability to search for flights available for a certain destination on a certain date. After finding and selecting the flight required, a user can book a ticket. Only registered users will be allowed to book a ticket; hence, while booking the ticket, the user has the option either to provide authentication details or, if not registered, to register with the application.

After either a valid authentication or a valid registration, the user will be prompted with a confirmation screen that displays the booking details and asks for a confirmation from the user. The final step for the user is to confirm the order; the application will then allocate a seat on that flight to the user.

The airline ticket booking system will have some major limitations because you wish to keep the application simple. A user can book only a single one-way ticket to the required destination. There will be no arrival date or arrival time displayed on the ticket. Also, even though the credit card details of the user are captured, these will not be used to generate any billing information.

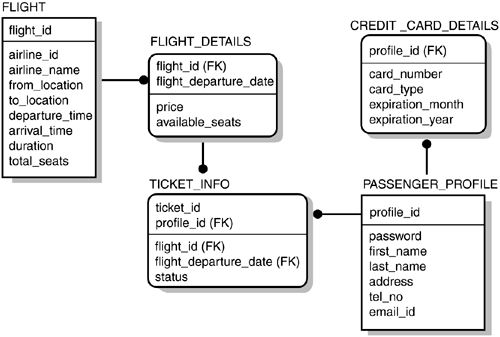
Application Design

You will be approaching the design of the airline ticket booking system in two ways: the database design and the class design. These two approaches are required because the application needs to persist data. Hence, the design discussion for the airline ticket booking system will include the Entity-Relationship (E-R) diagram and the class diagram.

E-R Diagram

Figure 2.5 shows the logical E-R diagram for the airline ticket booking system. After studying the requirements, you can identify the different entities involved in the application, such as the airline, passenger, flight, and ticket\_info. These entities will need to be normalized to establish proper relationships. Flights can have different departure dates; hence, the flight entity can be further refined into a flight entity and flight\_details entity. Because passengers purchase tickets for a flight, relationships are also defined between the passenger and ticket\_info entitites and the ticket\_info and flight entities.

Figure 2.5. Logical Entity-Relationship diagram for the airline ticket booking system.



Note

Normalized relationships in EJB 2.0 are not yet cleanly handled. Hence, if you need to define a relationship between two entities, EJB 2.0 mandates that the relationship columns between the two entities must be the same in terms of name and datatypes. For this MVC application, the columns of the airline and flight entities after normalization will not be the same, so you cannot use this level of normalization. For this reason, you will denormalize store redundant information of the airline entity in the flight entity. The application will not have the airline entity; unfortunately, you have to live with this limitation until the specification is updated.

Next you will look at the detailed database table design for the airline ticket booking system.

FLIGHT

The FLIGHT table contains information about the different flights offered by the airlines. This table is a master data table. The table structure is listed in Table 2.1.

| Table 2.1. FLIGHT Table Structure | | |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Remarks** |
| flight\_id | NUMBER | Represents the unique identifier for a flight of an airline. |
| airline\_id | NUMBER | Represents the unique identifier for an airline. |
| airline\_name | VARCHAR(20) | Represents the unique identifier for an airline. |
| from\_location | VARCHAR(20) | Represents the location from which the flight begins. |
| to\_location | VARCHAR(20) | Represents the destination of the flight. |
| departure\_time | DATETIME | Represents the time the flight departs every day. |
| arrival\_time | DATETIME | Represents the time the flight arrives every day. |
| duration | NUMBER | Represents the time taken for the flight. |
| total\_seats | NUMBER | Represents the number of seats on this flight. |

FLIGHT\_DETAILS

The FLIGHT\_DETAILS table contains information about the specific flights offered by the airlines. This table is a transaction table. The table structure is listed in Table 2.2.

| Table 2.2. FLIGHT\_DETAILS Table Structure | | |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Remarks** |
| flight\_id | NUMBER | Represents the unique identifier for a flight. This is a foreign key. |
| flight\_departure\_date | DATETIME | Represents the departure date for the flight. |
| price | NUMBER | Represents the price of the flight on a specific date. |
| available\_seats | NUMBER | Represents the seats available on a specific date for a flight. |

PASSENGER\_PROFILE

The PASSENGER\_PROFILE table contains information about the passengers registered with the application. This table is a transaction table. The table structure is listed in Table 2.3.

| Table 2.3. PASSENGER\_PROFILE Table Structure | | |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Remarks** |
| profile\_id | NUMBER | Represents the unique identifier for a passenger. This is a primary key. |
| password | VARCHAR(20) | Represents the password used by the passenger to authenticate with the application. |
| first\_name | VARCHAR(20) | Represents the first name of the passenger. |
| last\_name | VARCHAR(20) | Represents the last name of the passenger. |
| address | VARCHAR(20) | Represents the address of the passenger. |
| tel\_no | VARCHAR(20) | Represents the telephone number of the passenger. |
| email\_id | VARCHAR(20) | Represents the e-mail ID of the passenger. |

CREDIT\_CARD\_DETAILS

The CREDIT\_CARD\_DETAILS table contains information about the credit cards of a passenger registered with the application. This table is a transaction table. The table structure is listed in Table 2.4.

| Table 2.4. CREDIT\_CARD\_DETAILS Table Structure | | |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Remarks** |
| profile\_id | NUMBER | Represents the unique identifier for a passenger. This is a foreign key. |
| card\_number | VARCHAR(20) | Represents the credit card number used by the passenger to purchase tickets. |
| card\_type | VARCHAR(20) | Represents the credit card type used by the passenger. |
| expiration\_month | VARCHAR(2) | Represents the expiration month of the credit card of the passenger. |
| expiration\_year | VARCHAR(4) | Represents the expiration year of the credit card of the passenger. |

TICKET\_INFO

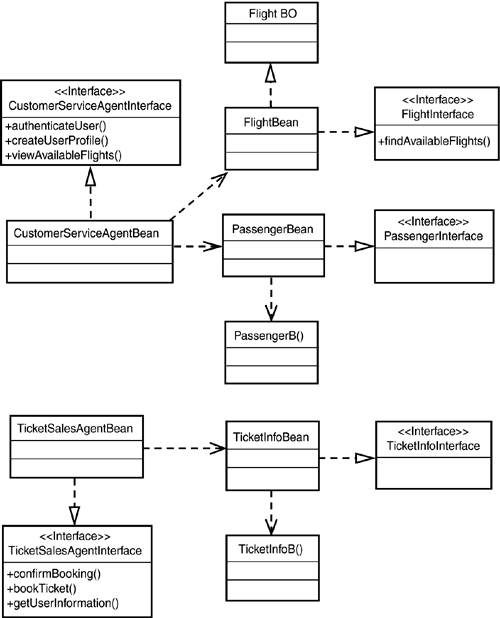
The TICKET\_INFO table contains information about the tickets purchased by the passenger. This table is a transaction table. The table structure is listed in Table 2.5.

| Table 2.5. TICKET\_INFO Table Structure | | |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Remarks** |
| ticket\_id | NUMBER | Represents the unique identifier for a ticket. |
| profile\_id | VARCHAR(8) | Represents the unique identifier for a passenger. This is a foreign key. |
| flight\_id | NUMBER | Represents the flight number of the flight booked by the passenger. This is a foreign key. |
| flight\_departure\_date | DATETIME | Represents the departure date of the flight booked by the passenger. |
| status | VARCHAR(10) | Represents the status of the ticket purchased by a passenger. |

Class Diagram

Figure 2.6 shows the class diagram for the airline ticket booking system. First, you will define the different entities in the application. From the database design, you already know about the airline, flight, passenger, and ticket entities. However, this will not suffice. You have to model your design on the Model View Controller architecture. Hence, you will organize your application classes in these layers and will require additional classes.

Figure 2.6. Class diagram for the airline ticket booking system.



Now, you will tackle the design layer by layer. You will cover a broad overview of the classes. The details of each class will be covered when you study each of the J2EE technologies supported by WebLogic Server.

Controller

You will have one servlet in your application that will perform the role of the Controller for the application. This will be the AirlineTicketBookingServlet class. All requests and responses between the browser and the rest of the airline ticket booking system will be routed through the AirlineTicketBookingServlet Java servlet. The AirlineTicketBooking Servlet class is organized in the com.sams.learnweblogic7.airlines.servlet package.

View

You will use JSPs for the different screens in your application. The JSPs will form the presentation layer for the application.

Model

Finally, the most important layer is the model layer, where the business logic and data logic for the airline ticket booking system application reside. The model layer consists of a set of EJBs. You will be using both session and entity beans in the application. You will take a look at the session beans first.

The user will interact via the Controller servlet with two session beans: CustomerServiceAgentBean and TicketSalesAgentBean. These are the two beans required to demarcate the two types of interactions that the user will have with the application. As long as the user performs interactions that do not need to have a session established with the application, the stateless session bean CustomerServiceAgentBean will process user requests. The CustomerServiceAgentBean is designed as a stateless session bean because it will process user requests, such as searching for available flights based on certain criteria, authenticating a user, and creating user profiles.

Once the user books a ticket and logs into the application, control is transferred to the stateful session bean TicketSalesAgentBean. The TicketSalesAgentBean provides services to book a ticket and retrieve the user's profile from the database. These business functions are performed by the session beans of the model layer. The CustomerServiceAgentBean and TicketSalesAgentBean classes are organized in the com.sams.learnweblogic7.airlines. sessionbean package.

The business logic part of the model layer interacts with the data logic part of the model layer. The data logic part of the model layer will be designed as a set of entity beans that will provide access to the required data for the airline ticket booking system application. The entity beans that you will design for the application are the FlightBean, PassengerBean, and TicketInfoBean. Each of these entity beans encapsulates a certain set of business data. The FlightBean provides the actual services to retrieve the available flights, which the CustomerServiceAgentBean session bean delegates to the FlightBean entity bean. The FlightBean entity bean encapsulates the result data of the search and provides methods to retrieve the result data. The PassengerBean entity bean encapsulates the data required for an authenticated user. The PassengerBean persists and retrieves the data associated with a users's profile registered with the application. The last entity bean that you will be designing for the application represents the ticket information after a user authenticates with the application and confirms the booking for a ticket. The TicketInfoBean provides a mechanism to create and retrieve the ticket information for a user.

The FlightBean, PassengerBean, and TicketInfoBean classes will be organized in the com.sams.learnweblogic7.airlines.entitybean package. You will be defining appropriate Home and Local interfaces for these entity beans.

Because the session beans will interact with the entity beans for processing and persisting the data flowing in the application, you will need to define a set of business objects that will be a replica of the entity beans they represent. These business objects are needed to ensure that the EJB entity bean classes and interfaces do not get directly referenced in the presentation layer or the MVCAppValueObject. The business objects are just plain Java classes that sufficiently isolate the entity bean data logic layer from the rest of the application. The three business object classes used are FlightDetails, PassengerProfile, and TicketDetails. They are grouped in the com.sams.learnweblogic7.airlines.businessobject package.

Support Classes

Because data will be passed back and forth across different layers in the MVC architecture of the airline ticket booking system application, you will define a MVCAppValueObject class, which will be used as a design contract between the model, view, and controller layers. The MVCAppValueObject is the only way that data will be consistently passed among the layers. This ensures that each of the layers can be sufficiently isolated from the others yet can interact with them. The MVCAppVAlueObject is a wrapper class over any data structure, normally a hash table.

The MVCAppValueObject class allows each of the layers to store data to be forwarded to the next layer for processing. For the constants of the application, you will define a MVCConstants class. For the fields that you will be using in the HTML forms in the View layer, you will define a JspFieldConstants class. The MVCAppValueObject, MVCConstants, and JspFieldConstants classes are organized in the com.sams.learnweblogic7.airlines.constants package.

You will build each of these components incrementally in the coming days.