Comp199 Applied Project

Albatross Travel Agency Website

Contributors: Candise Wang
Pasha Bolokhov
Toshiyasu Akazawa

Table of Content

Project Specification

❖ Overview

This web-site application represents a simplified travel company interface for selecting vacation trips around the world.

Users of the website can easily find vacation plans classified by regions on our website and purchase online through PayPal directly. Moreover, customers are allowed to create their own accounts for extra service.

❖ Application Requirement

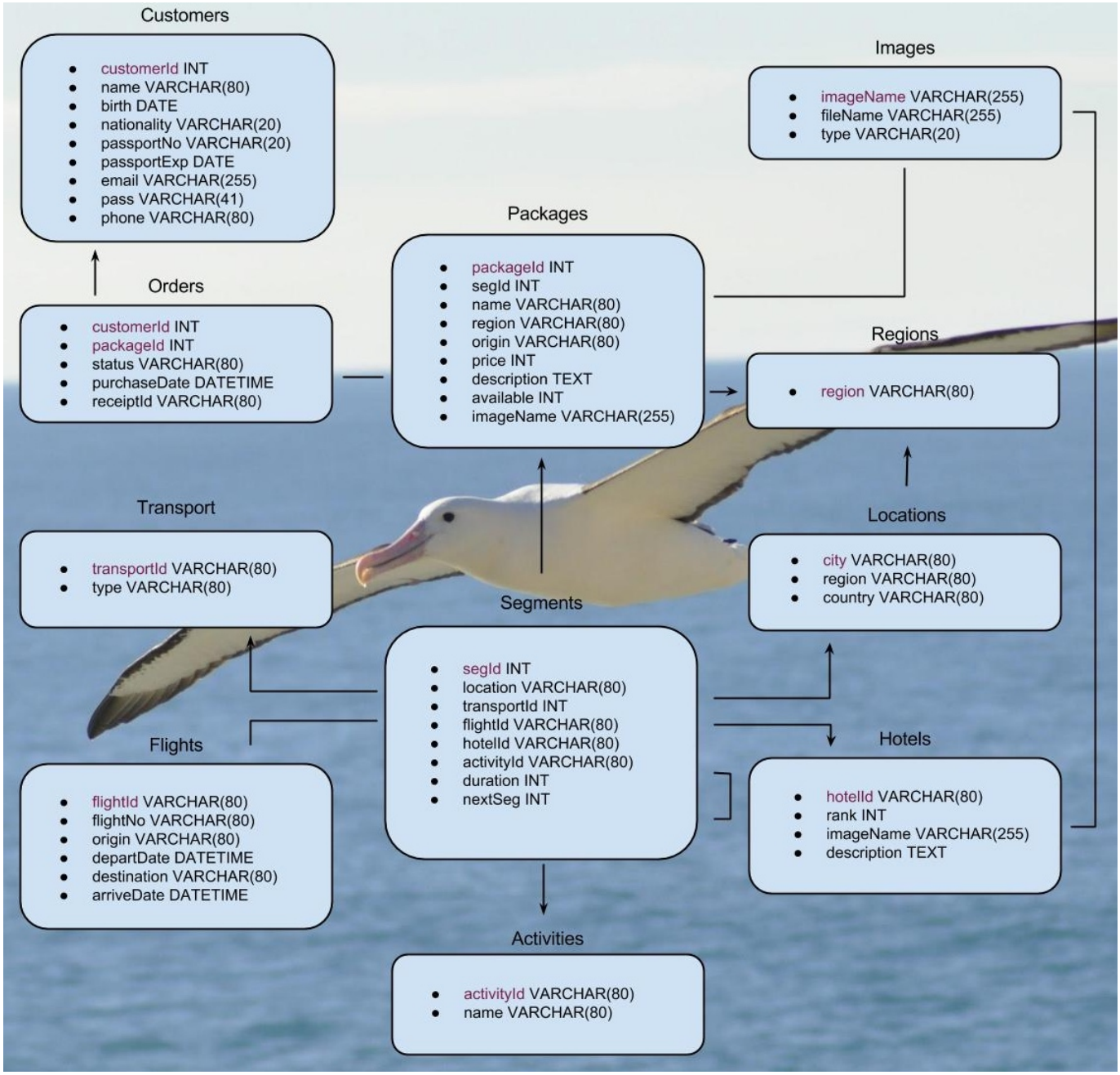
Functional requirements:

- A customer can register to create individual account (Prerequisite)
- A customer can modify the account information
- A customer can log in/log out
- Packages can be filtered by regions
- A vacation package can be added to order list
- A vacation package can be removed from order list
- A customer can pay through PayPal
- A customer can receive confirmation email after payment

Non-functional requirements:

- Using LAMP stack
- Using JWT tokens for authorization
- Other Technologies and libraries: Angular JS, Bootstrap, SPA, REST

❖ Entity Relationship Diagram Design



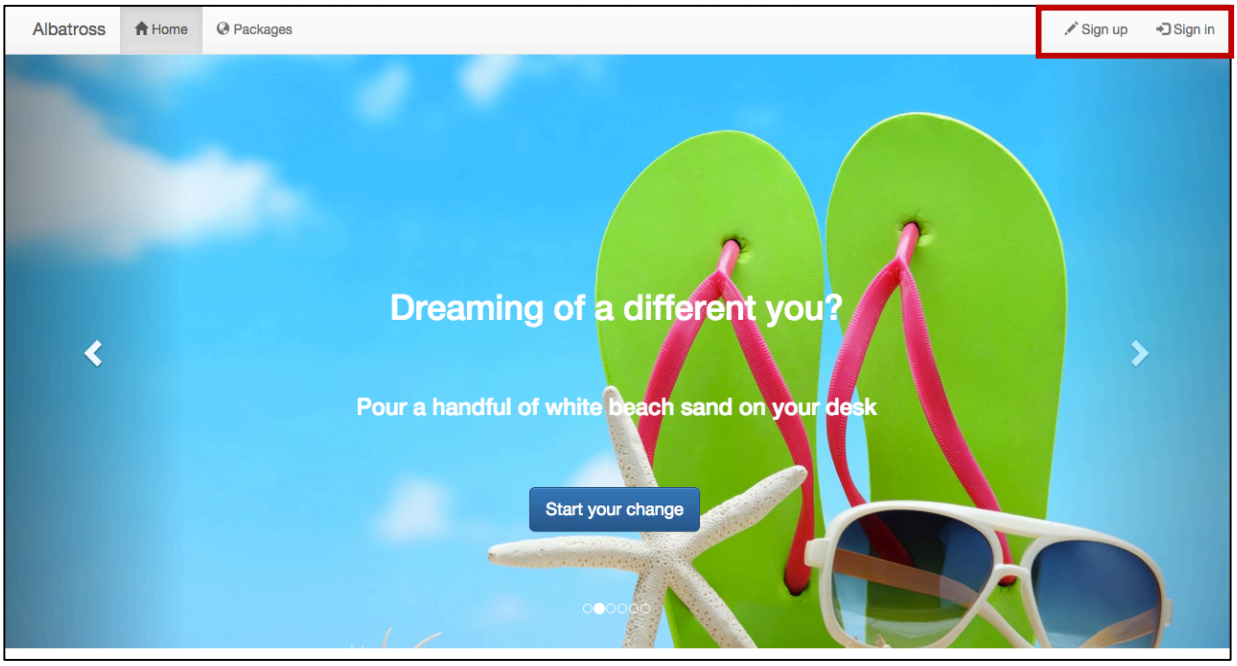
❖ Entity Relationship Diagram Design

Date	Designated Role	Detail Responsibility	Team Mamber		
			Toshi	Pasha	Candise
Week3	Team Leader	Weekly Report	x		
	Database/Back-end Develop	Create ERD	x	x	x
		Create tables and fields using MySQL	x	x	x
		Insert Sample Data			x
Week4	Team Leader	Weekly Report		x	
	Database/Back-end Develop	Continuous improvement	x		
	Middle-Wave Developer	access to the database		x	x
	Front-End Developer	Webpage design	x	x	x
		Home Page			x
		Navigation		x	
Week5	Team Leader	Weekly Report			x
	Database/Back-end Develop	Continuous improvement		x	
	Middle-Wave Developer	Sign Up	x	x	
		Sign In			x
	Front-End Developer	Account Information Page	x		
Week6	Team Leader	Weekly Report	x		
	Database/Back-end Develop	Continuous improvement			x
	Middle-Wave Developer	Display trip information		x	
	Front-End Developer	Pre-Packaged Trip Page		x	
Week7	Team Leader	Weekly Report			x
	Middle-Wave Developer	Options to search trips by “Region”		x	
		Generate/modifies trip orders	x	x	x
	Front-End Developer	Order management page		x	
Week8	Team Leader	Weekly Report			x
	Middle-Wave Developer	Payment using PayPal			x
		Display and modify account info	x		
	Front-End Developer	Upgrade the home page		x	
Week9	Team Leader	Weekly Report	x		
	Middle-Wave Developer	PayPal Redirection		x	
		Improve Profile interface	x		x
Week10	Team Leader	Weekly Report			x
	Middle-Wave Developer	Improve Profile interface	x		
		Send confirmation email			x
		Improve package details		x	

Usability Guide

❖ How to SignUp/SignIn?

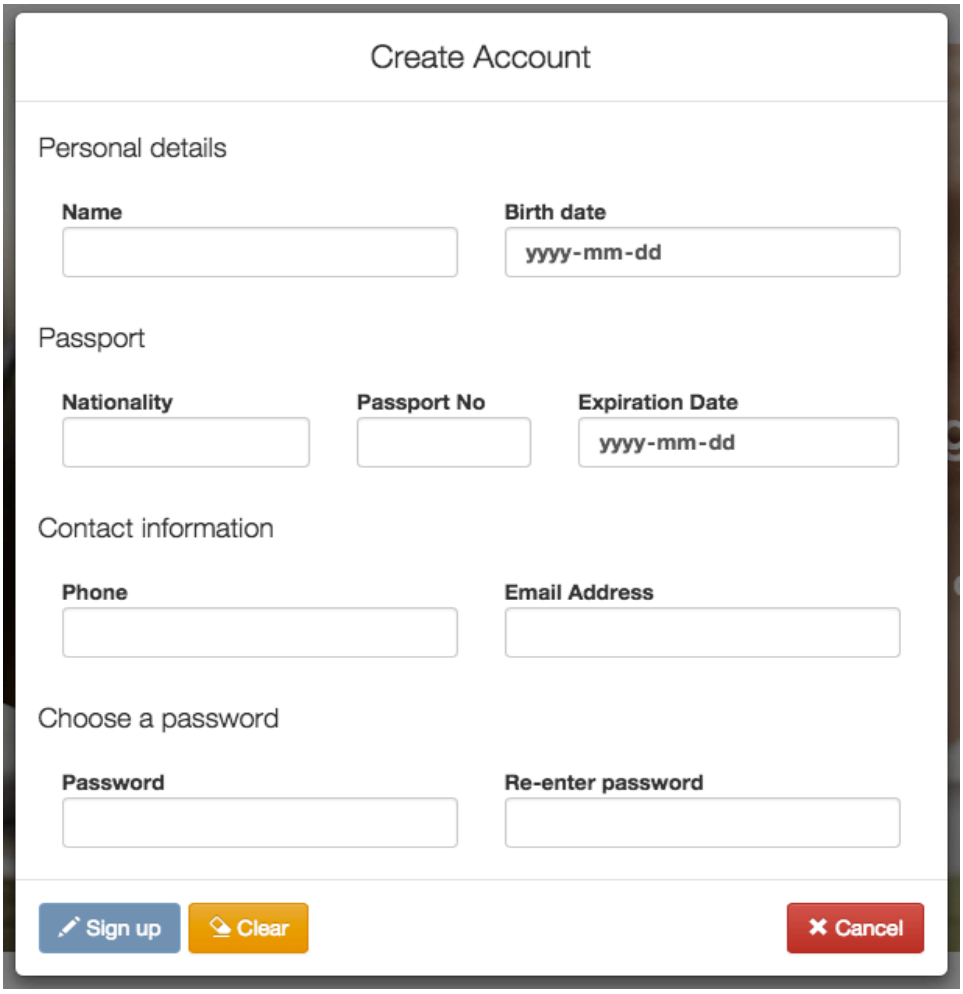
1. Launching at the homepage of Albatross Travel Agency, you will find the “SignUp” and “SignIn” button at the top right of the page.



2. Click on the “SignUp” button, you will see a popup modal asking for individual information to create a new account.

   Hit “Sign up” to complete the registration and navigate to trips information page.

   Hit “Clear” to reset the registration form.



3. If you already have an account, click the “Sign In” button. Enter your email and the password to complete the signing in and navigate to trips information page.

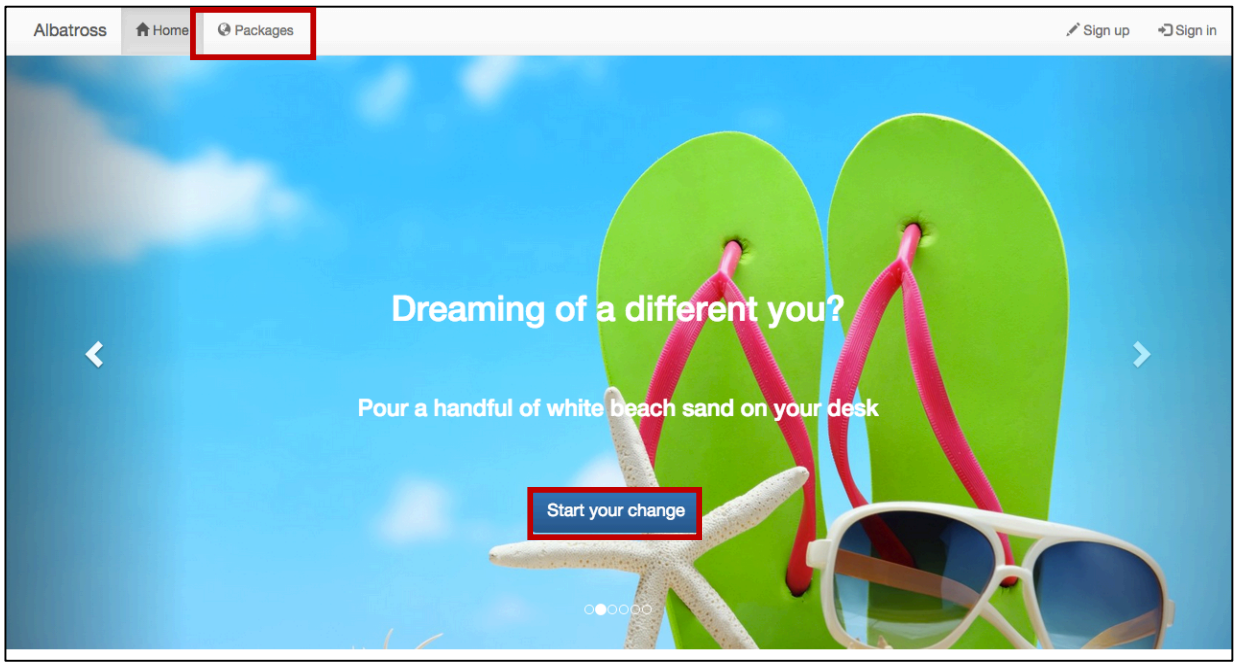
4. After Signing in, the navigation bar will change accordingly, providing more information and service.

   Left Navigation Bar:

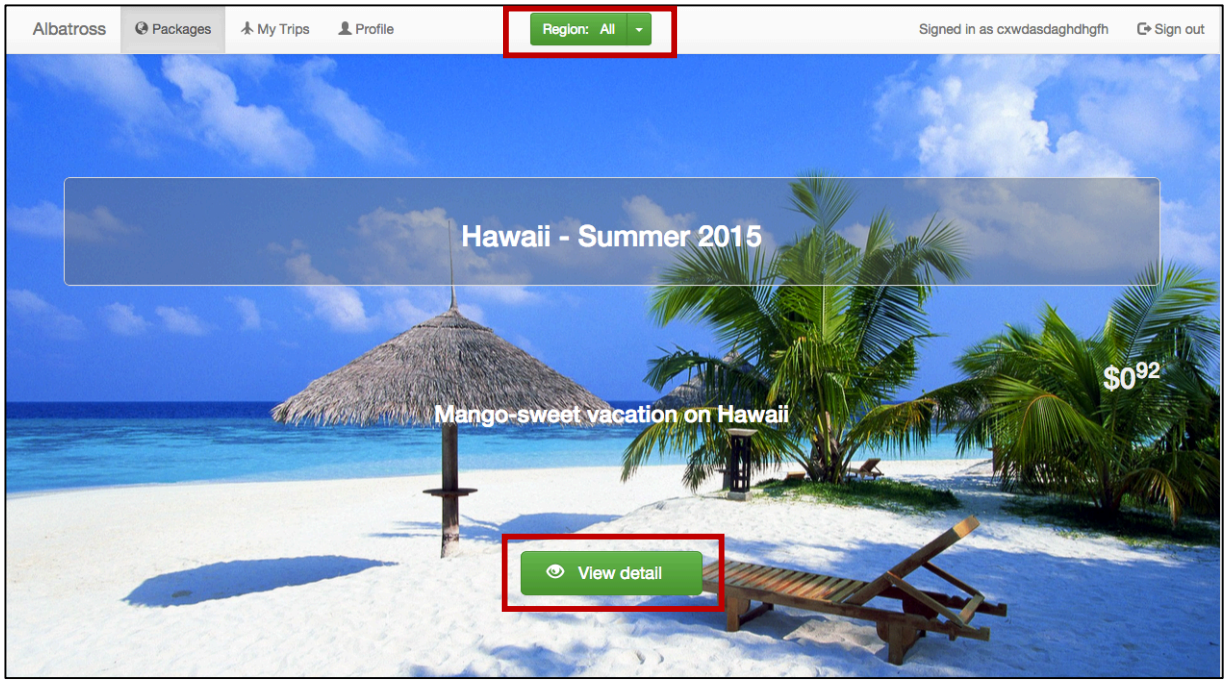
   Right Navigation Bar:

❖ How to search for a Trip?

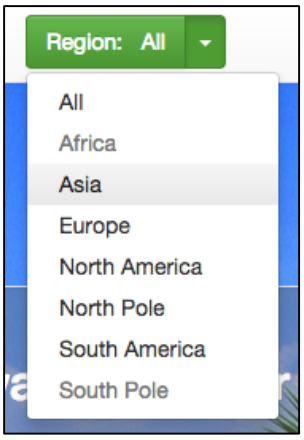
1. Click “Packages” on the navigation bar or click and button on the photo slider



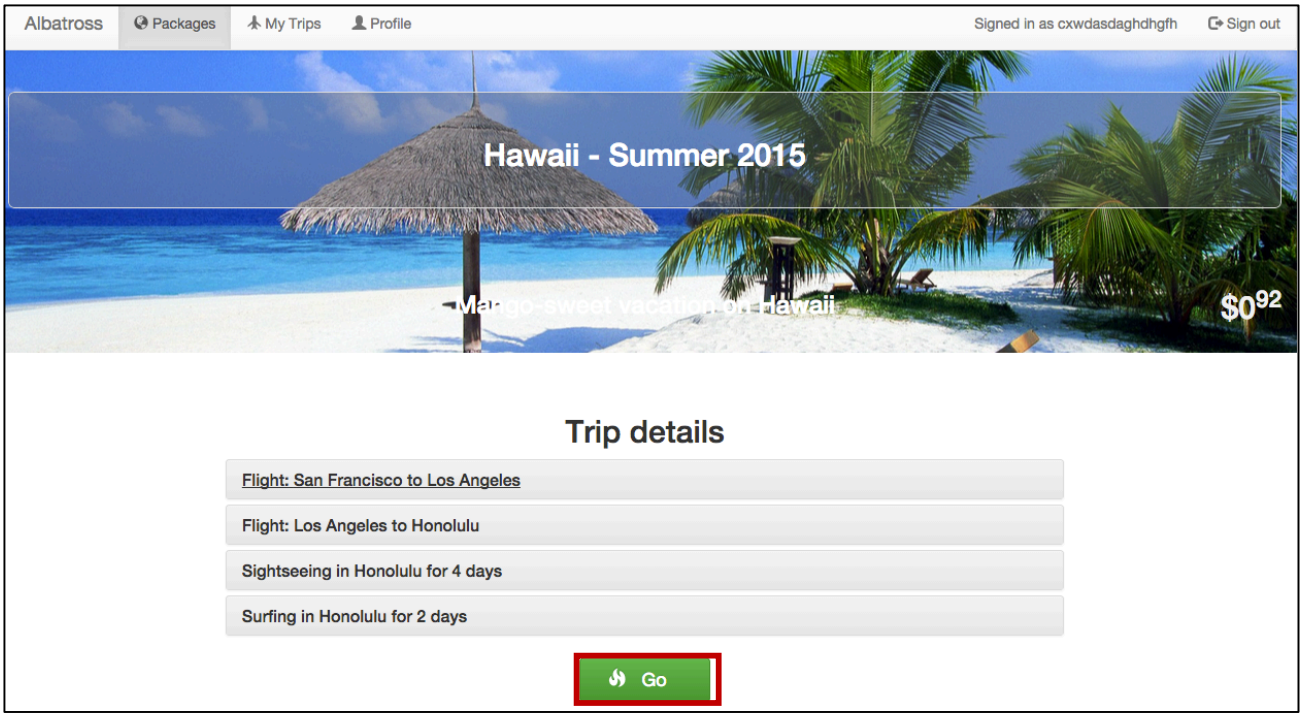
2. At “Packages” Page, you can see a list of vacation package with general information.



3. Click the dropdown filter “Region” on the top center of the navigation bar, you can select the preferred region for the vacation.



4. Click “View Detail” button on each package will lead you to the detail segments and schedule of the package.



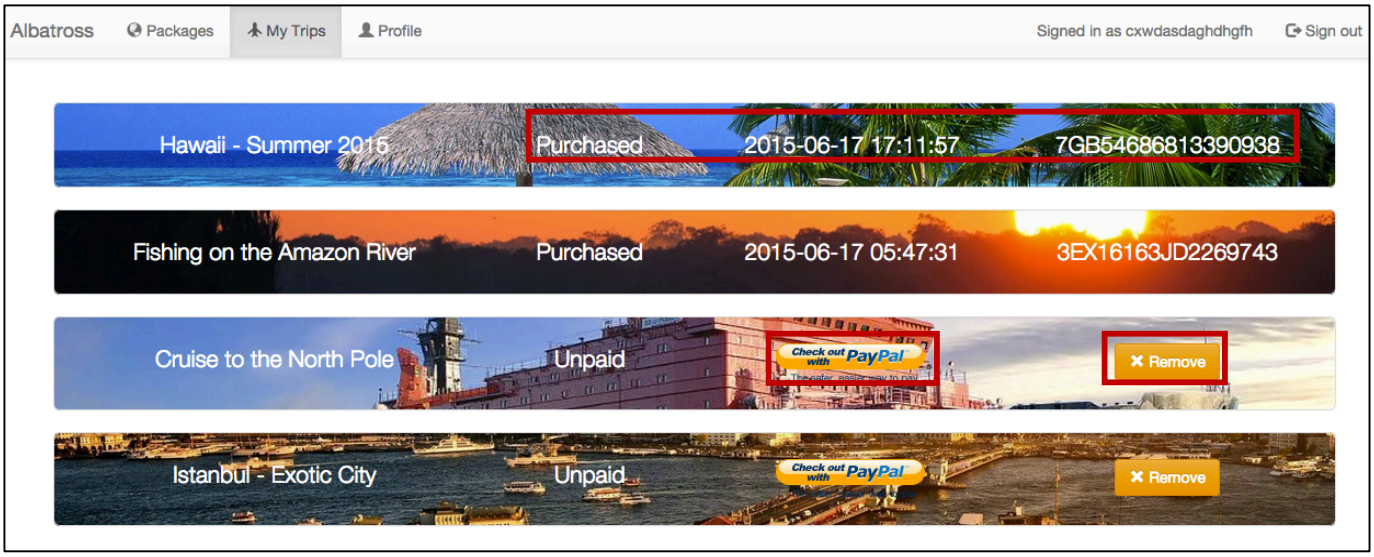
5. Click “Go” button in the Trip details page will add this vacation in your trip order list if you have already signed in. If not, you will be asked to sign in before next step.

❖ How to manage Trip Orders?

1. Click “My Trips” on the navigation bar. It will show a list of trip orders that you have been processed.

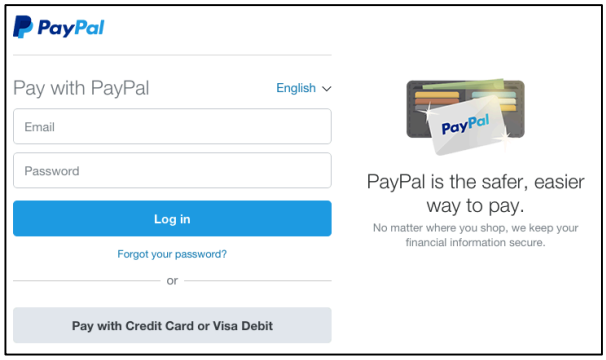
2. A paid trip will be labeled as “Purchased” with information of purchase date and receipt number.

An unpaid trip will be labeled as “Unpaid” with buttons to create payment or to cancel this trip order.

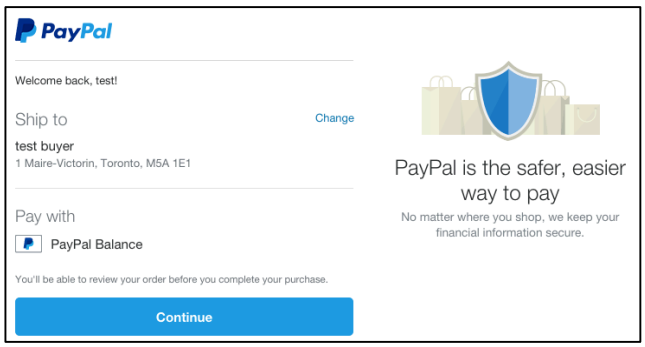


❖ How to pay for an order?

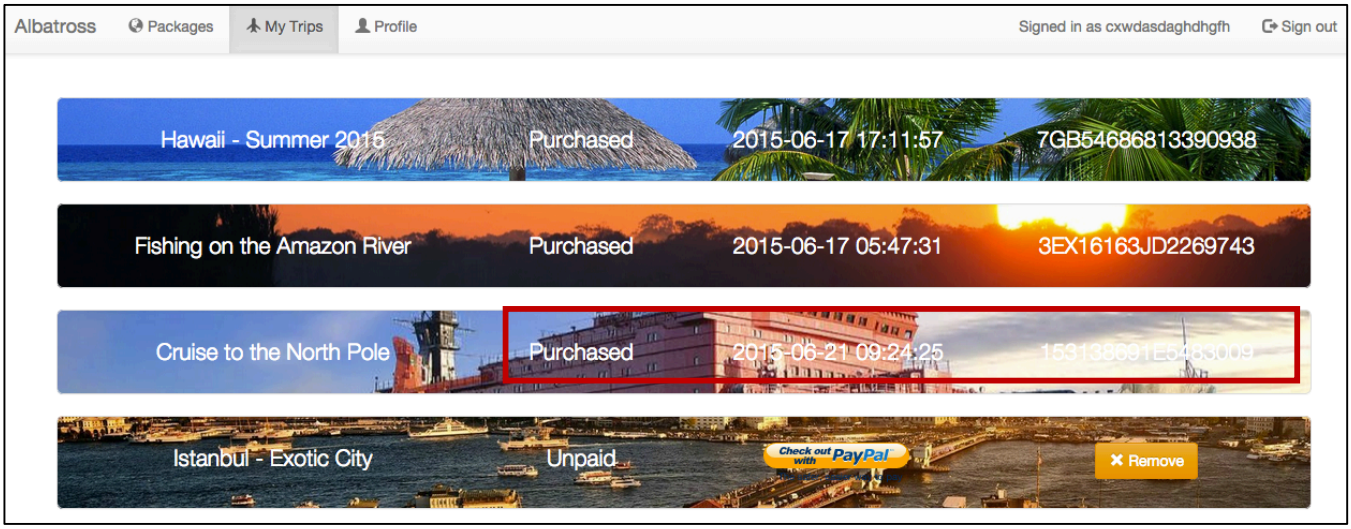
1. Click the “PayPal” Button to log into your PayPal account.



2. Click “Continue” to complete the payment.

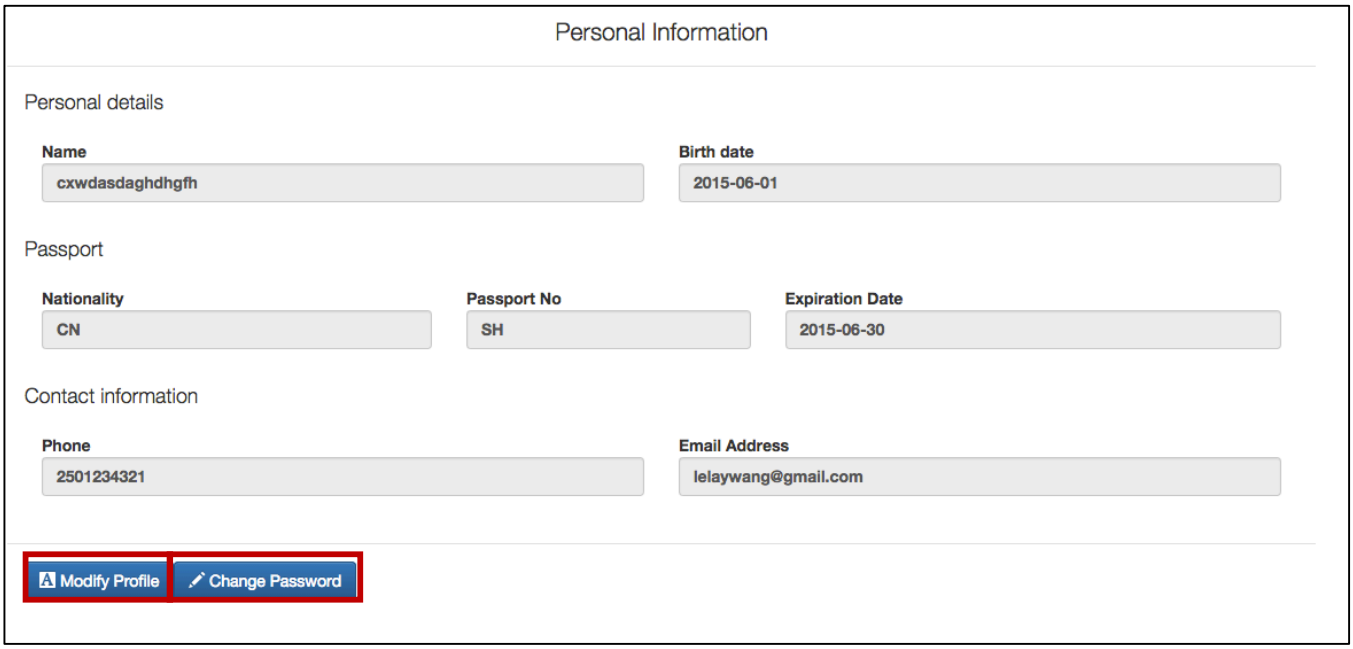


3. After paid successfully, the order list will automatically updated.



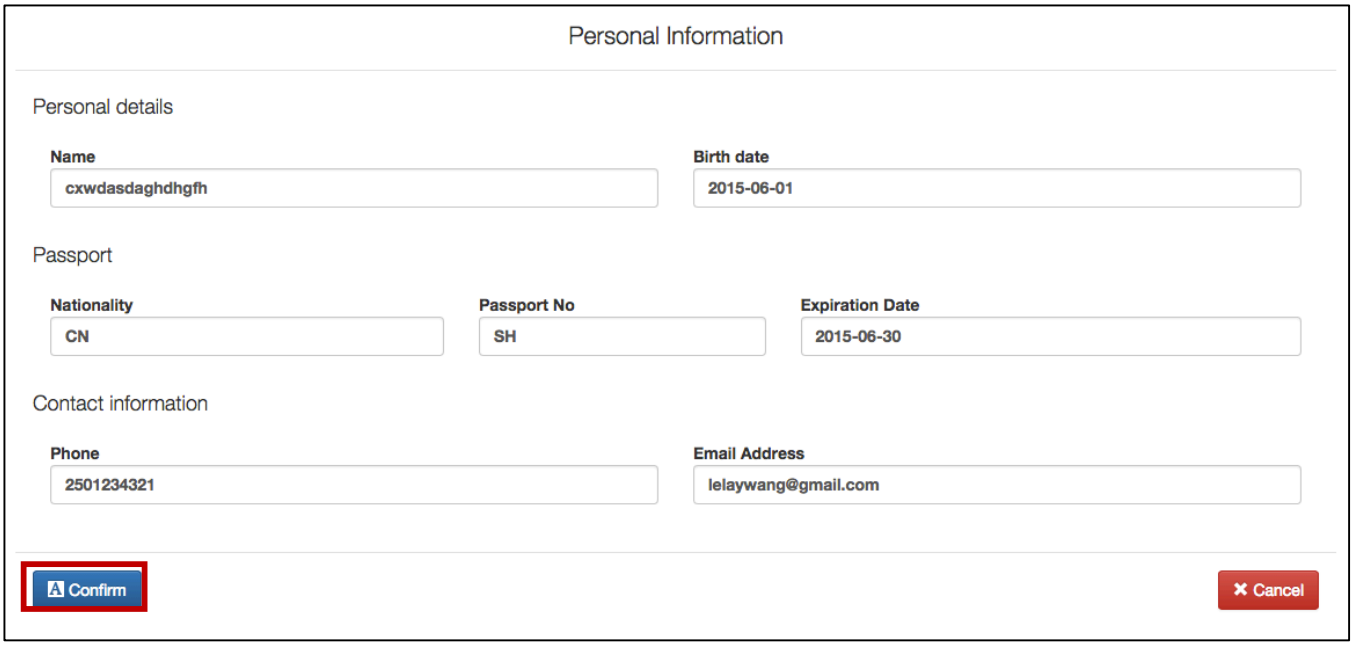
❖ How to manage account information?

1. Click “Profile” on the navigation bar to check the account information



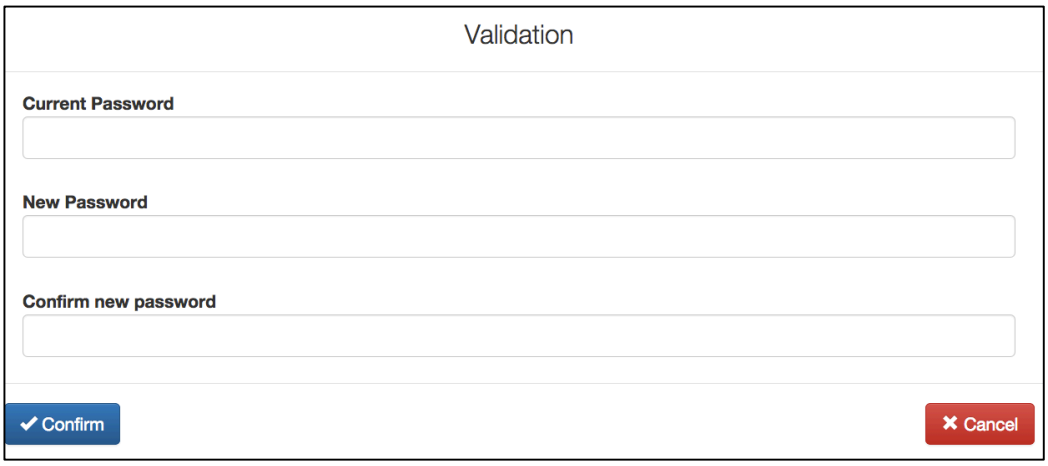
2. Click “Modify Profile” to update account information.

   Click “Confirm” to save the updates.



1. Click “Change Password” to reset password.

   Click “Confirm” to save the new password.



Learned Technologies

Single Page Application

In this project, we built this Single Page Applications using Angular JS, Bootstrap and etc. Single Page Applications are not application only consisting of one web-page. They are applications that are only loaded once — HTML, CSS and JavaScript are loaded together once. Extra data is uploaded later, when necessary, usually via Ajax. Most of web applications nowadays are SPA.

Remember how you had to include <html>, <stylesheet>, <script>, <DOCTYPE>, and all the menu items in your every PHP script? Every time the user clicks the menu, the browser has to load HTML, CSS, and all JavaScript scripts from scratch.

SPA are different. Like your bank account — it only is loaded once, but it uploads your account balance, profile and other information on request when you click around the menu. The web-page consists of many pages which sit in separate HTML “partials” — they have no headers, declarations — just HTML content (,). Since HTML is static, SPA needs Routing (menu) — to show you different things.

The application comprises a Front-end — a running JavaScript program. It connects to the server — Back-end. Reloading the app just to navigate to a “View recent bill” from the “Home” page is like rebooting the computer when you need to switch from one document to another in Microsoft Word.

There is no single permanently running Back-end program (besides the web-server itself, say Apache). We use PHP scripts (could be any language) to handle Ajax requests. Originally PHP was invented to include HTML mark-up. This creates a nightmare mess of the mixture of PHP code and HTML (some software e.g. Moodle still uses PHP mark-up for efficiency and portability). We run pure code on PHP and send JSON data to the App, which then decides how to update the mark-up (DOM).

Angular JS

Angular JS is a JavaScript library maintained by Google.
As of the day it is the most correct way of building an SPA, it completely isolates the user from handling DOM whatsoever:

- Extends HTML language to make it dynamic
- Two-way Data Binding, MVC (JavaScript variables hold all the data — user input, control, etc)
- HTML compilation, digest cycles
- Includes jqLite - a lighter version of JQuery for performance
- Includes $q - a lightweight implementation of Promises
- It's a framework too: 1000+ modules have been written for Angular (ngmodules.org)

Code Reference:

1. In HTML, we used directives "ng-repeat" instantiates a template once per item from a collection; "ng-show" shows or hides the given HTML element based on the given expression. Also, Angular JS expressions and data are bind inside double braces.

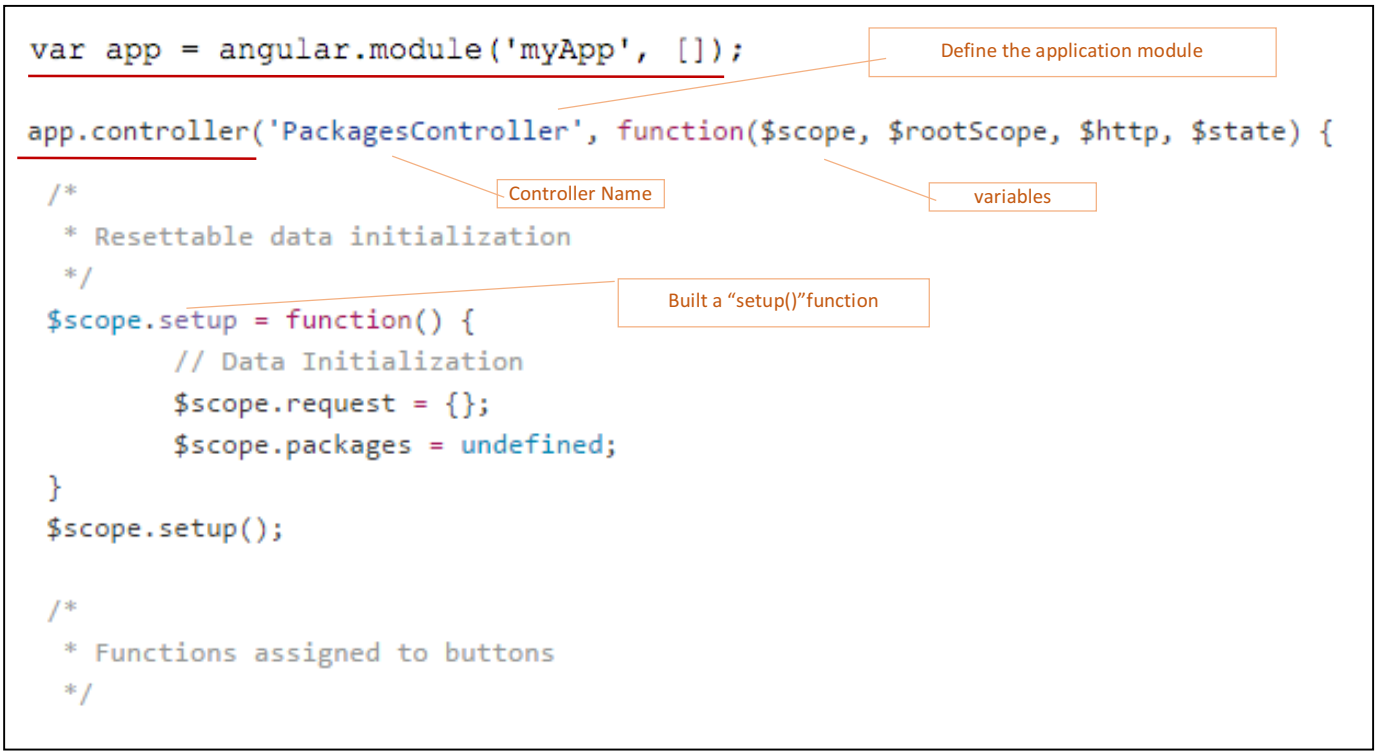
```
<div ng-repeat="p in packages" ng-show='p["available"] > 0'>

  <div ng-show='$index > 0'>
    <br><br><br><br><br><br>
  </div>

  <div class="jumbotron"
       style='background: url(images/{{ p["fileName"] }}) no-repeat center center;
              background-size: cover;
              margin-bottom: 0;'>

    <div class="container">
      <div class="row">
        <div class="col-xs-6">
          <br><br><br><br>
        </div>
      </div>
```

Directives

Directives

Data Binding

2. In JavaScript, we need to build a AngularJS module that define the application; then we can build controllers we need to control the application.



3. Below is the code we assigned to the "Region" filter button in the package page. we are using $http.post("php/packages.php") to call the PHP script and send the requests. $http.post will return a "promise", it is an object that has two important methods "success" and "error". So we invoke these two methods, and give them "functions" as parameters. These functions are called whenever the data has arrived (success) or there is something wrong (failure). It is in those functions where we do data processing.

```
$rootScope.regionSelect = function(r) {
        // Switch the current region to the selected value
        $rootScope.region = r;

        // Indicate we are waiting for data
        $rootScope.waiting = true;
        $scope.request = { region: $rootScope.region.region };

        // Send the request to the PHP script
        $http.post("php/packages.php", $scope.request)
        .success(function(data) {
                // process the response
                if (data["error"]) {
                        $rootScope.error = "Error: " + data["error"];
                } else {
                        $scope.packages = data["data"];
                }
        })
        .error(function(data, status) {
                console.log(data);
                $rootScope.error = "Error accessing the server: " + status + ".";
        })
        .finally(function() {
                // Indicate that we have an answer
                $rootScope.waiting = false;
        });
```

4. In PHP, it will get query from Json data, fetch information from database and send back the response.

```php
<?php
/**
 * This file fetches the packages from the database and
 * sends their information to the website
 *
 */
require_once 'validate.php';

/* Cancel very long responses */
define("MAX_RESPONSE_LINES", 1000);

/* get the query from JSON data */
$jsonData = file_get_contents("php://input");
$data = json_decode($jsonData);

/* validation */
if (!validate($data->region)) {
        $response["error"] = "Validation error";
        goto quit;
}

/* connect to the database */
require_once '../../../comp199-www/mysqli_auth.php';
$mysqli = @new mysqli(MYSQL_HOST, MYSQL_USER, MYSQL_PASS, MYSQL_DB);
if ($mysqli->connect_error) {
        $response["error"] = 'Connect Error (' . $mysqli->connect_errno . ') ' .
                             $mysqli->connect_error;
        goto quit;
}
```

UIRoute

Routing is used for implementing a menu in an SPA. Angular has a lightweight router — ngRoute module. The basic goal of the router is to put certain HTML content (a "partial" page) into a certain spot on the webpage, depending on which menu you click on. That content may itself be dynamic too (and usually it is).

In our case, we switched from the native ngRoute library to a more sophisticated AngularUI Router library for navigation. Rather than pages, it deals with states of the application. Each state specifies a rectangle spot on screen (or a few spots) and what HTML content to put there. Such a spot is called a view. When you click the menu items state transition occurs — those rectangles change their contents to another state (the native ngRoute library only allows for a single view)

```
 * User states
 */
.state("user.packagesRoot", { // This parent state just calls the controller
        url: "/packages",
        abstract: true
})
.state("user.packagesRoot.packages", {
        url: "",
        views: {
                "select-region-view@": {        // The view in the root state
                        templateUrl: "partials/regions.html"
                },
                "@": {          // Targets the unnamed view in the root state
                        templateUrl: "partials/packages.html",
                        controller: "PackagesController"
                }
        }
})
.state("user.packagesRoot.packages.view", {
        url: "/view",
        params: {
                package: null
        },
        views: {
                "select-region-view@": {        // The view in the root state
                        // Do not show the region select tool
                },
                "@": {          // Targets the unnamed view in the root state
                        templateUrl: "partials/packages.view.html",
                        controller: "PackagesViewController"
                }
        }
```

Define state with names under $stateProvider

"Abstract: true" indicate this is a parent state

All HTMLs under the views will display in this state

Assign controller to specific html file

Defined a state parameter which will be transferred or injected when state changes

Below is a function assigned to a button using “go()” method to direct to different state.

```
$scope.view = function(name) {
        $state.go('.view', { package: name });
};
```

Json Web Token

Authentication is one of the most important parts of any web application. For decades, cookies and server-based authentication were the easiest solution. However, handling authentication in modern Mobile and Single Page Applications can be tricky, and demand a better approach.
Normally the (back-end) server would maintain the logged-in/out status of the customer. This has difficulties with scalability -- if you have a 100 of servers, they need a mechanism to sync sessions. The best known solution to authentication problems used in Web applications is the JSON Web Token (JWT). PayPal uses JWT as well. JWT allows to build a *RESTful* application.

A token is a simple Json structure which contains customer’s login name, optionally role (user/admin), and (also optionally) expiration time.

{ "iss": "Albatross Travel","name": "Katy Perry", "admin": false, “exp”: 1434540890 }

It also contains an authentication header, with a cryptographic signature. The user is unable to alter the token without screwing up the signature. Anyone who has a token is granted access. Privacy of the token in the real world is ensured by HTTPS. No session information is stored on the server. No other login process.

```
/**
 * This function generates a JWT token based on $name and $email
 *
 * @params      $name           Full customer's name
 *              $email          Customer's email
 *
 * @returns                     A JWT token
 *
 */
function generate_jwt($name, $email) {
        $token = array(
                "iss"   =>      "Albatross Travel",
                "iat"   =>      time(),
                "name"  =>      $name,
                "email" =>      $email
        );
        return JWT::encode($token, JWT_KEY, 'HS256');
}
```

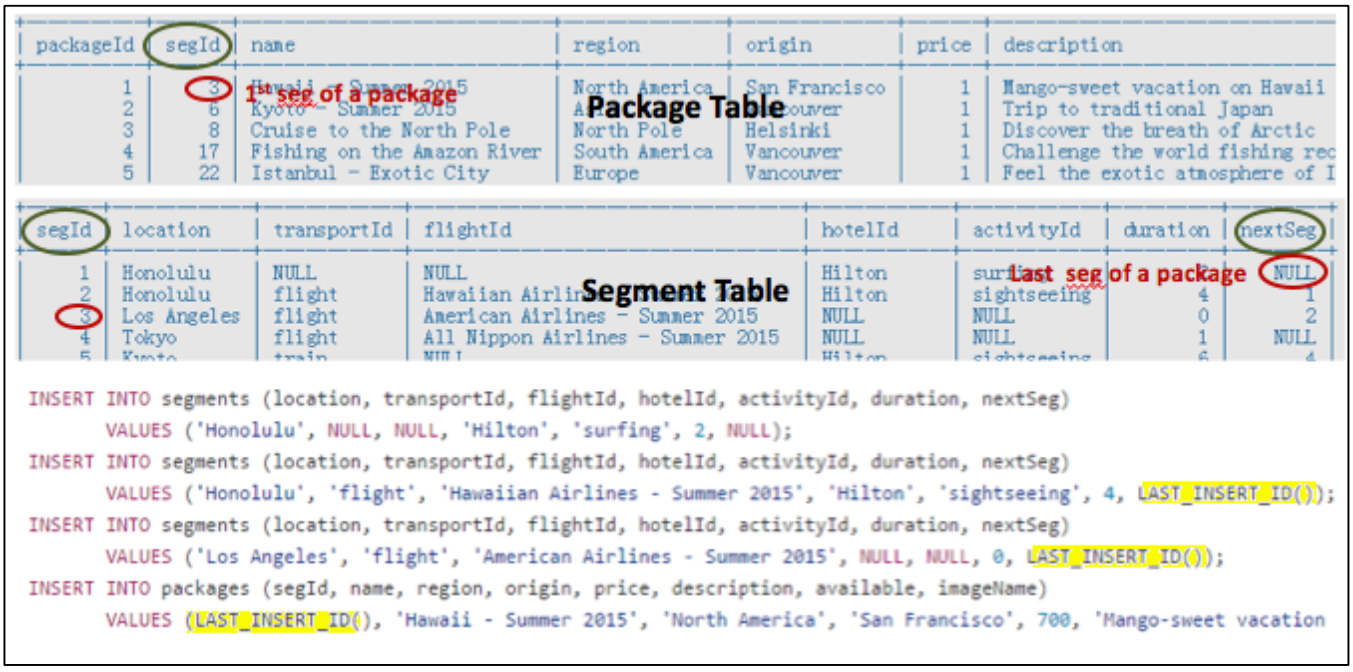
Problems and Solution

❖ Database Design

1. Simplifying our data structure and the product complexity: We removed individual flight and hotel reservation requirement and decided to focus on implement only pre-packaged trips and other main functionality.

2. Dealing with the overlap/ redundant trip information (such as activities) between similar vacation packages: We created a package table for storing brief description, start place and capacity, and another segment table for detail information of difference destination.

   There is a “nextSeg” column in each entry that indicates what is the next destination within the same package(the last destination will have “null” value)



❖ Navigation

During the implementation of the interface, we want that a “guest” customer could be only accessible to home and package page. The customer will be forced to sign in as a “user” to add orders or reach other private information.

In order to better handle this, rather than having lots of boolean check for each behavior, we switched from the native ngRoute library (cannot trace a "sequence" of states) to a more sophisticated AngularUI Router library for navigation. This was dictated by the presence of multiple phases or states of the app, eg: logged in / logged out, packages → orders → check-out.

Another important change was switching the authentication to JWT tokens instead of using PHP cookie-based sessions. The token authentication mechanism dramatically simplifies handling of all requests in the per-customer domain.

❖ PayPal Redirection

We use "In-Context Checkout Experience" — when the user clicks "Pay with PayPal" button, a new small window pops up which navigates to PayPal for the customer to authorize the transaction. The Web-App stays active in the background. This is implemented in PayPal's small library "checkout.js"

However, the problem is that after the user confirms the transaction, this library redirects the original App to a different "confirmation" page. The confirmation URL includes the PayerId and PaymentId for the app to send to server.

Moreover, when attaching "Pay" buttons to multiple products on a page, PayPal does not really provide any feedback which button was chosen by the client.

To handle these unexpected problems with a SAP, we modified the "checkout.js" library so that it uses a callback which it calls in the case of the user's consent for transaction. This is instead of redirection.

```
/* place an order */
$scope.pay = function(event, idx) {                ← function called when a button is clicked

    /* Initialize PayPal environment */
    paypal.checkout.initXO();

    /* Send request to the server */
    $scope.request = { package: $scope.unpaidTrips[idx] };    ← choose the selected product from the array
    $rootScope.waiting = true;
    $http.post("php/secure/create-payment.php", $scope.request)    ← ask server to call PayPal to create payment
    .success(function(data) {
        // process the response
        if (data["error"]) {
            if (data["error"] == "authentication")
                $rootScope.doSignOut();
            else
                $rootScope.error = "Error: " + data["error"];

            paypal.checkout.closeFlow();
            return;
        }

        // success
        if (!data["ec_token"]) {
            $rootScope.error = "Error: no EC-token received";
            paypal.checkout.closeFlow();
            return;
        }
        $scope.ecToken = data["ec_token"];

        paypal.checkout.startFlow($scope.ecToken);          ← open PayPal window for user to confirm
    })
    .error(function(data, status) {
        console.log(data);
        $rootScope.error = "Error accessing the server: " + status + ".";
```

```
paypal.checkout.closeFlow();
    })


    .finally(function() {
        $rootScope.waiting = false;
    });
};

/* order accepted */
$scope.accepted = function(url) {          ← function called by PayPal when the user authorizes the transaction
    $scope.paymentRequest = {};
    $scope.paymentRequest.url = url;    ← the URL contains unique combination of PayerId and PaymentId
    $rootScope.waiting = true;
    $http.post("php/secure/execute-payment.php", $scope.paymentRequest)    ← ask server to execute payment
    .success(function(data) {
        // process the response
        if (data["error"])
            switch (data["error"]) {
            case "email-failed":
                $rootScope.error = "payment went through but ";
                $rootScope.error += "couldn't send email to " + $rootScope.storage.token.email;
                break;        // consider it a success still

            case "authentication":
                $rootScope.doSignOut();
                return;

            default:
                $rootScope.error = "Error: " + data["error"];
                return;
            }

        // success - refresh the list of trips
        $scope.getOrders();          ← on success the server automatically updates the list of products
```

Conclusion

❖ Further Improvement

With sufficient time, we could make the application capable to provide:

- Detailed information/receipt page for each plan/purchase
- Options for choosing the quantity of each trip
- More complicated services and functions such as DIY trip package
- Options for quick payment without signing in

❖ Valuable Learned

- How to create a Single Page Application
- Using AngularJS data binding features
- Using UI-Route for navigation
- Using Json Web Token to save log-in state on the client-side
- Using Bootstrap (one of very popular libraries for making responsive content and easy web-page formatting)
- Using GIT version control application

Besides the all the techniques we learned to build the application, we also experienced how to work out in a team environment. For example, learning from teammates and helping each other when someone stuck at some problem; managing time and deal with changes or delay compare to our planned schedule. After all, we are proud of our final delivery and thanks to everyone that helped and contributed during the project.