

Delivery planning

Scenario

A wholesale company distributes outdoor activity equipment to sports shops. Goods are supplied by van to customers throughout Wales from a distribution depot in Newtown in mid-Wales. An on-line software system is required to record orders, then to plan the most efficient routes for deliveries.



All users of the system will be staff of the wholesale company. For the purpose of this example project, it is assumed that orders from sports shops will be received by e-mail or phone and will be entered into the computer system by the office staff. Direct on-line ordering has not been included in the project, but this function is left as an additional programming exercise if required.

Facilities will be required for entering the details of products in stock, including photographs. The trade customers must be registered and their delivery locations recorded. Orders will then be entered.

The procedure for planning a delivery journey begins by displaying a list of orders awaiting delivery. Staff can select the orders from this list which will be included in the delivery journey, perhaps taking into account the availability of stock or the capacity of the delivery van. The computer will then determine the shortest circular route to visit each of the customers and return to the depot. A delivery schedule will be displayed and the mileages between delivery points calculated.

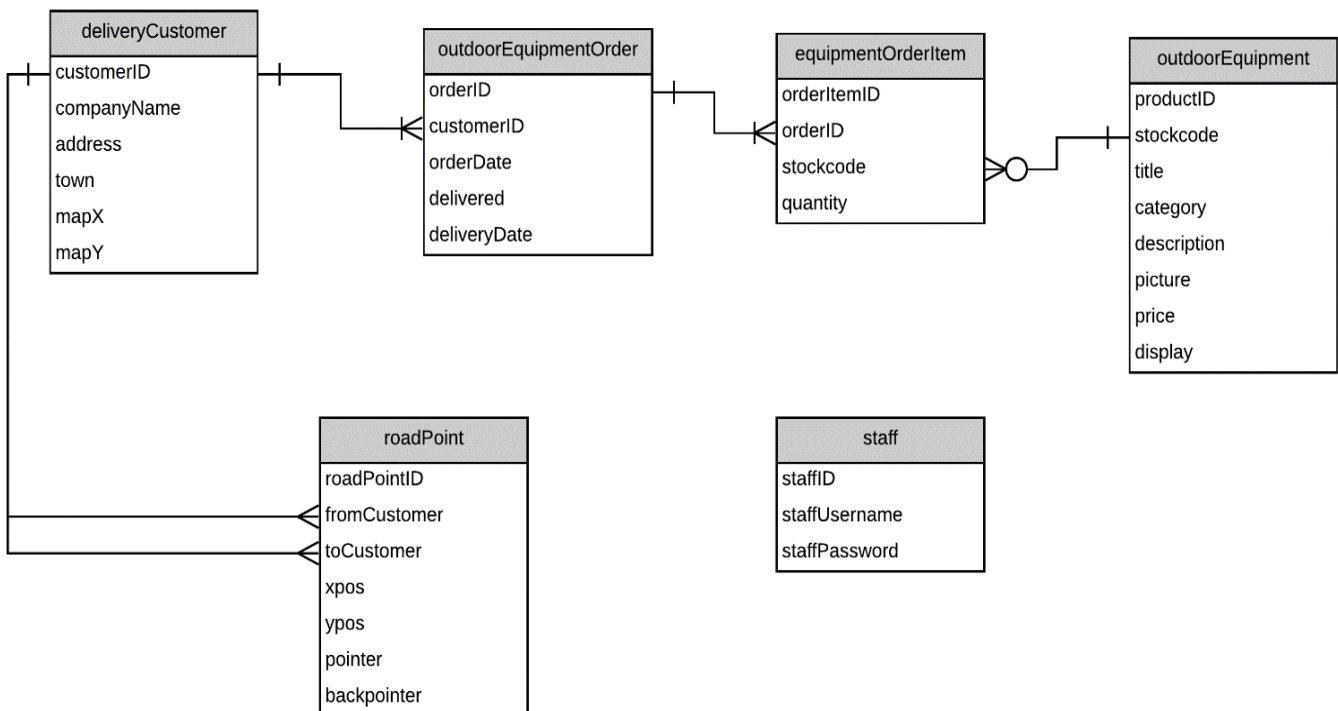
It is appreciated that routes between customers will not be straight lines, but may follow the geography of valleys or the coastline. Facilities should be available for creating realistic routes.

Design

The system will be designed around a series of linked database tables:

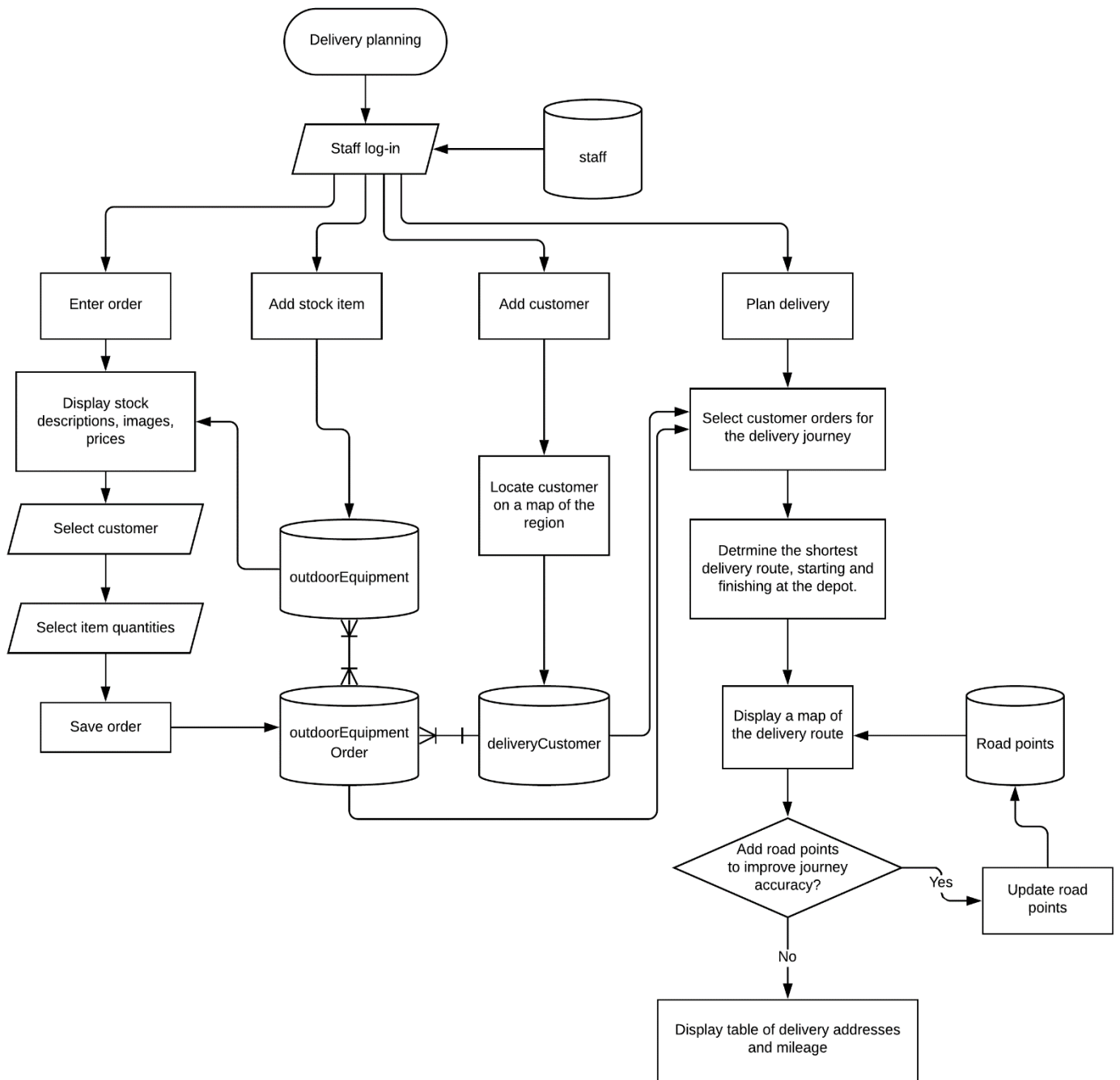
Trade customers are registered and their business name and address are entered in the **customer** table. In addition, the (x,y) map coordinates of the delivery address are recorded.

Orders received are noted in the **order** table. Each order is allocated a unique **orderID**, and is linked by **customerID** to the corresponding customer record.



Programming techniques

The project uses PHP with objects to handle data from the database tables. JavaScript with the p5.js high level language extension is used for the map applications, locating customers, displaying and editing the delivery route. The overall structure of the project is illustrated by the flowchart below.



The website will be designed as a series of pages, each with a specific function:

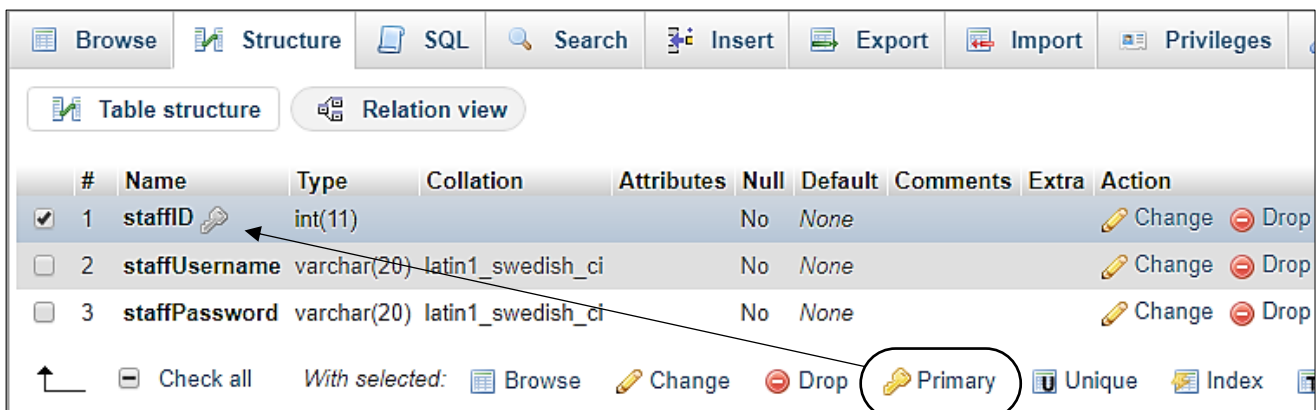
- Staff log-in by entering a user name and password, which will be verified against records in the database **staff** table.
- A page allows information for each product to be entered, which is then stored in the **outdoorEquipment** table. The product record will include the file name for a photograph image, which will be stored separately in an **uploads** folder on the server.
- Customer records can be added to the **deliveryCustomer** table. In addition to entering customer contact information, the input page will provide a road map of the delivery region on which the customer's location can be pin-pointed. This will be stored as X-Y map coordinates for use in determining the delivery route.

- A web page allows orders to be entered. The customer is first selected. Products can then be chosen from a list, with the required quantities specified. An option allows the full description of a product to be displayed, along with its photograph. Overall order details will be stored in the **outdoorEquipmentOrder** table, linked to a set of individual product orders in the **equipmentOrderItem** table.
- When planning a delivery, a list of outstanding orders is first displayed. The orders to be included in the delivery are then selected.
- The program uses the nearest neighbour algorithm to determine a minimum journey distance starting at the depot and travelling around the series of delivery points. The algorithm is repeated using each delivery point in turn as the starting point of the circuit. A shorter total distance may be found. The best result is chosen as the journey route and displayed on a map of the region.
- Interactive graphics allow the user to add points along the road connection between delivery points, to allow the link to more accurately follow the route and reflect the true distance travelled. The sequence of road points between locations is stored in the **roadPoints** table in the form of linked lists.
- A table lists the sequence of delivery addresses, along with the calculated mileages for the stages of the delivery route.

Method

Begin by setting up a new folder on your local computer and on the server with the name '**delivery**'.

All users will be staff of the company, so a password login is required. Log-in to the PHP MyAdmin web site for your database account and display the list of tables in the database. Select the **New** option from the list of tables. Set up a database table for staff usernames and passwords. Create three fields: **staffID** as integer, **staffUsername** and **staffPassword** both of type varchar with a length of 20 characters. Name the table as '**staff**' and save the table design.



Set the **staffID** field to be the primary key. Click the Change option on the **staffID** line, then tick the auto increment (**A_I**) box. Further information about setting up the staff table will be found in the Hardware Store project in Chapter 2.

Use the **Insert** option to add several members of staff as test data.

Create a header image for the homepage with a size of approximately 1000 pixels by 240 pixels. This should include the company name 'Outdoor Equipment Wholesale'.



Save the graphics file as **title.jpg** and copy it to the **delivery** folder on the server.

Open a blank file. Create the log-in page by entering the program code below.

```
<?
  session_start();
  $_SESSION['login']='NO';
?>
<html>
<head>
  <title>Delivery planning</title>
  <style>
    body{
      font-family: arial, sans-serif;
    }
  </style>
</head>
<body>
  <form action="enterOrder.php" method="post">
  <img src='title.jpg'>
  <table border="0" cellpadding="10">
  <tr>
    <td><h3>Staff Log-in</h3></td></tr>
  <tr>
    <td>User name</td>
    <td>
      <?
        echo "<input type=text size=20 name=user >";
      ?>
    </td></tr>
  <tr>
    <td>Password</td>
    <td>
      <?
        echo "<input type=password size=20 name=pass >";
      ?>
    </td></tr>
  <tr>
    <td></td>
    <td>
      <input type=submit value="Enter">
    </td></tr>
  </table>
  </form>
</body>
</html>
```

Save the file as **index.php** and copy it to the server. Run the website and check that the log-in page appears correctly. Password entry should be shown as symbols.

Staff Log-in

User name

Password

We will now create a **Staff** class. Open a blank file and add the lines of program code below.

```

<?
class Staff
{
    private $user;
    private $pass;
    function __construct($userSet,$passSet)
    {
        $this->user = $userSet;
        $this->pass = $passSet;
    }
    private function checkUser($userWanted,$passWanted)
    {
        if (($userWanted==$this->user)&&($passWanted==$this->pass))
            return true;
        else
            return false;
    }
    public static function checkPassword($userWanted,$passWanted)
    {
        include ('user.inc');
        $conn = new mysqli(localhost, $username, $password, $database);
        if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
        $query="SELECT * FROM staff";
        $result=mysqli_query($conn, $query);
        $num=mysqli_num_rows($result);
        mysqli_close($conn);
        $i=1;
        while ($i <= $num)
        {
            $row=mysqli_fetch_assoc($result);
            $user=$row["staffUsername"];
            $pass=$row["staffPassword"];
            $staff[$i] = new Staff($user,$pass);
            $i++;
        }
        $found=false;
        for ($i=1;$i<=$num;$i++)
        {
            $answer= $staff[$i]->checkUser($userWanted,$passWanted);
            if ($answer==true)
            {
                $found=true;
            }
        }
        return $found;
    }
}
?>

```

Save the file as **Staff.php** and copy it to the server. The program code defines username and password attributes for a staff object, and provide a constructor method. We then add methods to check the log-in details. The **checkPassword()** method will loop through all of the staff objects, applying the **checkUser()** method to each. If any of the individual objects returns a true result for a correct log-in, the **checkPassword()** method will return a true result overall.

All pages within the staff section of the web site will display the same menu options along the top of the page. We can save repetition by creating a separate file for the menu program code, which can be included in each staff page.

Enter order	Plan delivery	Add customer	Add stock item
-------------	---------------	--------------	----------------

Open a blank file. Add the program code shown below, then save this as **staffMenu.php**. Copy the file to the server.

```
<table class=menu>
  <tr><th class=menu>
    <a href="enterOrder.php">
      Enter order</th>
  <th class=menu>
    <a href="deliveries.php">
      Plan delivery</a></th>
  <th class=menu>
    <a href="addCustomer.php">
      Add customer</a></th>
  <th class=menu>
    <a href="addStockItem.php">
      Add stock item</a></th></tr>
</table>
```

Open a blank text file and set up a style sheet with the lines of code shown in the box below.

```
body {
  font-family: arial, sans-serif;
}
table.menu {
  border-collapse: collapse;
  width: 100%;
}
th.menu {
  text-align: left; padding: 8px;
  background-color: rgb(0, 153, 216);
  color: white;
}
a:link, a:visited {
  color: white;
  text-decoration: none;
}
#entryForm {
  width: 900px;
  margin: 20px; padding: 20px;
  border: 1px solid #000000;
}
```

Save the file as **styleSheet.css** and copy it to the server. We now have all the components necessary to create the first page within the web site. Open a blank file and add the lines of program code below.

```

<?
    session_start();
    $user=$_REQUEST['user'];
    $pass=$_REQUEST['pass'];
    $login=$_SESSION['login'];
?>
<html>
<head>
    <title> Delivery route </title>
    <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
    <?
    if (!($_SESSION['login']=='YES'))
    {
        include('Staff.php');
        if (Staff::checkPassword($user,$pass)==false)
            header('Location: index.php');
        else
            $_SESSION['login']='YES';
    }
    include('staffMenu.php');
    ?>
</body>
</html>

```

Save the file as **enterOrder.php** and copy it to the server. Before running the staff log-in system, a security file will be needed to authorise access to the on-line database. This has the format:

```


<?
    $username="YOUR USER NAME";
    $password="YOUR PASSWORD";
    $database="YOUR DATABASE NAME";
?>

```

Create a blank text file and copy the lines above. Replace "YOUR USER NAME" and "YOUR PASSWORD" with the username and password which give you access to the PHP MyAdmin website. The entry for "YOUR DATABASE NAME" is normally the same as the username entered on the first line. Save the small file as **user.inc** and copy it to the server.

Run the website. Enter a correct staff username and password. The page containing the staff menu should be displayed. Test the log-in system by entering incorrect details. The user should be returned directly to the log-in page.

The series of menu options are displayed at the top of the page. We will begin by developing the option to **enter products**. The product data will be stored in the database. Go to the PHP MyAdmin website and list the tables in the database. Select the 'new' option and create a table with the name **outdoorEquipment**. Add the fields as shown below. The **productID** field is of *integer* data type, and should be set to auto-increment as records are added.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	productID 	int(11)			No	None		AUTO_INCREMENT
2	stockcode	varchar(20)	latin1_swedish_ci		No	None		
3	title	varchar(50)	latin1_swedish_ci		No	None		
4	category	varchar(50)	latin1_swedish_ci		No	None		
5	description	varchar(500)	latin1_swedish_ci		No	None		
6	picture	varchar(50)	latin1_swedish_ci		No	None		
7	price	double			No	None		

We will now set up a web page to enter product details. Open a new file and add the lines of program code below. Save the file as **addStockItem.php** and copy it to the server.

```
<html>
<head>
  <title> Delivery route </title>
  <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
  <?
    include('staffMenu.php');
  ?>
  <div id="entryForm">
  <form action="upload.php" name="itemEntryForm" method="post"
    enctype="multipart/form-data">
  <table bgcolor='white' border='0' cellpadding='10'>
  <tr><td><h3>Add stock item</h3></td></tr>
  <tr><td><table bgcolor='white' border='0' cellpadding='10'></td></tr>
  <tr><td>Stock code</td>
  <?
    echo"<td><input type='text' name='txtStockcode' id='stockcode'
      width='300px' value='".$txtStockcode."></td>";
  ?></tr>
  <tr><td>Product title</td>
  <?
    echo"<td><input type='text' name='txtTitle' id='title' width='300px'
      value='".$txtTitle."></td>";
  ?></tr>
  </table>
  </td></tr></table>
  </form>
  </div>
</body>
</html>
```

Run the website. Log-in as a member of staff, then select the **'Add stock item'** menu option. Check that input boxes have been created for the stock code and product title.

Enter order	Plan delivery	Add customer
Add stock item		
Stock code	<input type="text"/>	
Product title	<input type="text"/>	

Return to the **addStockItem.php** file and add the lines of program code on the next page.

Save the **addStockItem.php** file, copy it to the server and refresh the **'Add stock item'** web page. A drop-down list of outdoor equipment categories should appear, as in the illustration below.

Category	<div style="border: 1px solid black; padding: 2px;"> <div style="background-color: #e0e0e0; padding: 2px;">Clothing, boots, rucksacks ▾</div> <div style="background-color: #e0e0e0; padding: 2px;">Clothing, boots, rucksacks</div> <div style="padding: 2px;">Tents and camping</div> <div style="padding: 2px;">Mountain bikes</div> <div style="padding: 2px;">Water sports</div> <div style="padding: 2px;">Climbing equipment</div> </div>
----------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```

<td>Product title</td>
  <?
    echo"<td><input type='text' name='txtTitle' id='title' width='300px'
      value='".$txtTitle."'></td>";
  ?></tr>
  <tr>
    <td>Category</td>
    <td>
      <select name="lstCategory" id="category">
        <?
          $cat[1]='clothing';      $text[1]='Clothing, boots, rucksacks';
          $cat[2]='camping';      $text[2]='Tents and camping';
          $cat[3]='bikes';        $text[3]='Mountain bikes';
          $cat[4]='watersports';  $text[4]='Water sports';
          $cat[5]='climbing';    $text[5]='Climbing equipment';
          for ($i=1; $i<=5; $i++)
          {
            if ($lstCategory==$cat[$i])
              echo"<option value='".$cat[$i]."' selected>".$text[$i];
            else
              echo"<option value='".$cat[$i]."'>".$text[$i];
            echo"</option>";
          }
        ?>
      </select></td></tr>
    </table>
  </td></tr></table>

```

Return to the **addStockItem.php** file and add the lines of program code below.

```

  ?>
  </select></td></tr>
  <tr>
    <td>Description</td>
    <?
      echo"<td>";
      echo"<textarea rows = '6' cols = '29' name = 'txtDescription'
        id='description'>$txtDescription</textarea>";
      echo"</td>";
    ?></tr>
  <tr>
    <td>Price £</td>
    <?
      echo"<td><input type='text' name='txtPrice' id='price' width='100px'
        value='".$number_format($txtPrice,2)."'></td>";
    ?></tr>
  </table>
</td></tr></table>
</form>
</div>
</body>

```

Continuing to work on the **addStockItem.php** file, add the further block of program code:

```

        <td>Price £</td>
        <?
            echo"<td><input type='text' name='txtPrice' id='price' width='100px'
                value='".number_format($txtPrice,2)."'></td>";
        ?></tr>
    </table>

    </td>
    <td valign='top'>
        <table border="0" cellpadding="10">
            <tr>
                <td>Image</td>
                <td><input type="file" name="fileToUpload" id="fileToUpload">
            </tr>
            <tr>
                <td><input type="submit" value="Upload Image" name="submit"><p>
                <?
                    echo"<img src='uploads/$imageFile' width='200'>";
                    echo"<input type='hidden' value='". $imageFile."' id='imageFile'>";
                ?></td>
            </tr>
        </table>

    </td></tr></table>
</form>
</div>
</body>

```

The code adds input components for the product description, price, and a photograph image.

Save the updated **addStockItem.php** file, then copy it to the server. Run the web site, go to the **addStockItem** page and check that components are displayed correctly as shown below.

Add stock item

Stock code <input style="width: 100%;" type="text"/>	Image <input type="button" value="Choose file"/> No file chosen
Product title <input style="width: 100%;" type="text"/>	<input type="button" value="Upload Image"/>
Category <input style="width: 100%;" type="text" value="Power tools ▼"/>	
Description <div style="border: 1px solid gray; height: 40px; width: 100%; margin-top: 5px;"></div>	
Price £ <input style="width: 100%;" type="text" value="0.00"/>	

The image upload will be carried out by PHP program code in another file. To create this, open a blank file and add the program code shown on the page below. Save the file as **upload.php** and copy it to the server.

The program begins by collecting data from the input boxes on the **AddStockItem** page, then re-saves this data as session variables for later re-use. The selected image file is copied to an uploads folder on the server.

```

<?
    session_start();
    $txtStockcode = $_REQUEST["txtStockcode"];
    $txtTitle=$_REQUEST['txtTitle'];
    $lstCategory=$_REQUEST['lstCategory'];
    $txtDescription=$_REQUEST['txtDescription'];
    $txtPrice=$_REQUEST['txtPrice'];
    $imageFile=basename($_FILES["fileToUpload"]["name"]);

    $_SESSION["imageFile"]= $imageFile;
    $_SESSION["txtStockcode"] = $txtStockcode;
    $_SESSION["txtTitle"]= $txtTitle;
    $_SESSION["lstCategory"]= $lstCategory;
    $_SESSION["txtDescription"]= $txtDescription;
    $_SESSION["txtPrice"]= $txtPrice;

    $target_dir = "uploads/";
    $target_file = $target_dir.$imageFile;
    move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file);
    header('Location: addStockItem.php?imageUploaded=YES');
?>

```

Create a folder with the name '**uploads**' within the 'delivery' folder on the server.

When we arrive back at the addStockItem page, any data previously entered in the input boxes should be redisplayed. To do this, open the **addStockItem.php** file and add the block of PHP code at the start as shown below.

```

<?
    session_start();
    $imageUploaded=$_REQUEST['imageUploaded'];
    if ($imageUploaded=='YES')
    {
        $imageFile=$_SESSION['imageFile'];
        $txtStockcode=$_SESSION["txtStockcode"];
        $txtTitle=$_SESSION["txtTitle"];
        $lstCategory=$_SESSION["lstCategory"];
        $txtDescription=$_SESSION["txtDescription"];
        $txtPrice=$_SESSION["txtPrice"];
    }
?>

<html>
<head>
    <title> Delivery route </title>
    <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>

```

Save the **addStockItem.php** file and copy it to the server. Run the website and select the '**Add stock item**' menu option. Enter a stock code, product title, description and price for an item of outdoor equipment, and select a suitable product category. Obtain a photograph image, save it onto your computer, then select this image file using the '**Choose file**' component. Finally, click the '**Upload image**' button. Check that the photograph and all text data are still displayed correctly after the page is reloaded.

The product record will be stored in the database by means of the **OutdoorEquipment** class which we will now create.

Add stock item

Stock code


Product title

Category

Description

Price £

Image No file chosen



Open a blank file and add the lines of program code below.

```
<?
class OutdoorEquipment
{
    public static $product= array();
    private $productID;
    private $stockCode;
    private $category;
    private $title;
    private $description;
    private $picture;
    private $price;

    function __construct($productID, $stockCode, $category,$title, $description,
                        $picture, $price)
    {
        $this->productID = $productID;
        $this->stockCode = $stockCode;
        $this->category = $category;
        $this->title = $title;
        $this->description = $description;
        $this->picture = $picture;
        $this->price = $price;
    }

    public static function saveRecord($productID, $stockCode, $category,$title,
                                     $description, $picture, $price)
    {
        include('user.inc');
        $conn = new mysqli(localhost, $username, $password, $database);
        if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
        $query="INSERT INTO outdoorEquipment VALUES ('','$stockCode', '$title',
            '$category','$description', '$picture','$price') ";
        echo $query;
        $result=mysqli_query($conn, $query);
        mysqli_close($conn);
    }
}
?>
```

Save the file as **OutdoorEquipment.php** and copy it to the server. The lines of code define the attributes for an OutdoorEquipment object, then provide a constructor method. Finally, a **saveRecord()** method is added to upload records to the outdoorEquipment database table.

Re-open the **addStockItem.php** file and add the lines of program code below. Save the updated file and copy it to the server.

```

</form>

<center>
<p>
<button type="button" onclick="button_click()">Save record</button>
</center>
<script>
function button_click()
{
    stockcode = document.getElementById("stockcode").value;
    title = document.getElementById("title").value;
    category = document.getElementById("category").value;
    description = document.getElementById("description").value;
    description = description.trim();
    imageFile = document.getElementById("imageFile").value;
    price = document.getElementById("price").value;
    destination = 'saveStockItem.php?stockcode='+ stockcode+'&title='+title
                  + '&category=' + category + '&description='+ description + '&price='
                  + price + '&imagefile=' + imageFile;
    window.location.href= destination;
}
</script>
</div>
</body>
</html>

```

Open a new file and add the code below. Save the file as **saveStockItem.php** and copy it to the server.

```

<html>
<head>
<?
    $stockcode=$_REQUEST["stockcode"];
    $title=$_REQUEST["title"];
    $category=$_REQUEST["category"];
    $description=$_REQUEST["description"];
    $imagefile=$_REQUEST['imagefile'];
    $price=$_REQUEST["price"];
    ?>
</head>
<body>
<?
    include('OutdoorEquipment.php');
    OutdoorEquipment::saveRecord('', $stockcode, $category, $title,
    $description, $imagefile, $price);
    header('Location: enterOrder.php');
    ?>
</body>
</html>

```

This block of code begins by collecting the field values from the page URL. The `saveRecord()` method in the `StockItem` class is then called to transfer the record into the database. When saving is completed, the program will return to the blank page where the staff menu is displayed.

Log-in to the web site as a member of staff. Go to the `addStockItem` page, insert details of a product including a picture, and save the record. Check that the image has been uploaded to the `uploads` folder on the server, and that the product record appears in the `outdoorEquipment` table of the database. If all is working correctly, add a series of products to the database.

productID	stockcode	title	category	description	picture	price
1	AP5613	Alpine jacket	clothing	Extremely versatile and lightweight. Perfect for c...	jacket.jpg	130
2	FW6792	Light hiking boot	clothing	Made from extremely high quality durable materials...	boot.jpg	32.6
3	TN7856	Backpacking tent	camping	Tent for solo backpackers and campers, providing f...	tent1.jpg	65
4	CA6782	Sleeping bag	camping	Insulation designed to enhance thermal capacity an...	sleepingBag.jpg	36.5
5	EQ6781	Climbing helmet	climbing	Moulded EPS foam with low profile polycarbonate sh...	helmet.jpg	36.9
6	WS567	Canadian canoe	watersports	Fiberglass construction for strength and durabilit...	canoe1.jpg	488.3
7	MB672	Mountain bike	watersports	Alloy mountain bike frame with steel suspension fo...	bike1.jpg	163.5
8	CN872	Kayak	watersports	This kayak offers the paddler one boat that does i...	kayak.jpg	398.4
9	TR9122	Two-person expedition tent	camping	A lightweight but robust two-person tent suitable ...	tent2.jpg	89.5

We can now work on the page which will input customer orders. Extra entries will be needed in the style sheet file which will format the screen display.

Open the `styleSheet.css` file and add the entries shown below. Save the file and copy it to the server.

```
table.stock {
    border-collapse: collapse;
    border: 1px solid gray;
}
tr.stock {
    border: 1px solid gray;
}
td.stock, th.stock {
    padding: 5px;
    border: 1px solid gray;
}
```

Before setting up the page to input customer orders, we must add some extra methods to the `OutdoorEquipment` class. A `loadStockItems()` method will access product records from the database and create a set of objects. The attributes of these objects can then be displayed on the web page using a series of `get()` methods.

Open the `OutdoorEquipment.php` file and add the program code shown below. Save the file and copy it to the server.

```

public static function loadStockItems()
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM outdoorEquipment";
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    $i=1;
    while ($i <= $num)
    {
        $row=mysqli_fetch_assoc($result);
        $productID=$row["productID"];
        $stockCode=$row["stockcode"];
        $title=$row["title"];
        $category=$row["category"];
        $description=$row["description"];
        $image=$row["picture"];
        $price=$row["price"];
        $obj = new OutdoorEquipment($productID, $stockCode, $category,
                                   $title, $description, $image, $price);
        OutdoorEquipment::$product[$i] = $obj;
        $i++;
    }
    return $num;
}

public function getProductID(){return $this->productID;}
public function getStockCode(){return $this->stockCode;}
public function getTitle(){return $this->title;}
public function getCategory(){return $this->category;}
public function getDescription(){return $this->description;}
public function getImage(){return $this->picture;}
public function getPrice(){return $this->price;}
}
?>

```

We can now return to work on the page where customer orders will be input. Open the file **enterOrder.php** and add the lines of program code shown in the two boxes below. Save the file and copy it to the server. The program produces a table of stock items and prices.

```

<?
    if (!($_SESSION['login']=='YES'))
    {
        include('Staff.php');
        if (Staff::checkPassword($user,$pass)==false)
            header('Location: index.php');
        else
            $_SESSION['login']='YES';
    }
    include('staffMenu.php');

    include('OutdoorEquipment.php');
    $itemcount= OutdoorEquipment::loadStockItems();

?>
</body>
</html>

```



```

include('staffMenu.php');
include('OutdoorEquipment.php');
$itemcount= OutdoorEquipment::loadStockItems();
?>
<p>
<table cellpadding=10>
<tr><td>
  <table class=stock cellspacing=10px>
  <col width="800">
  <col width="100">
  <col width="100">
  <tr class=stock>
    <th class=stock > Stock item</th>
    <th class=stock > Wholesale price</th>
    <th class=stock >Quantity required</th>
  </tr>
  <?
  for ($i=1;$i<=$itemcount;$i++)
  {
    echo"<tr class=stock><td class=stock >";
    $code=OutdoorEquipment::$product[$i]->getStockCode();
    echo $code;
    $stockcodes[$i]=$code;
    echo": ".OutdoorEquipment::$product[$i]->getTitle();
    echo"</td>";
    $price=OutdoorEquipment::$product[$i]->getPrice();
    echo"<td class=stock>£".number_format($price,2) ."</td>";
  }
  $_SESSION['stockcodes']=$stockcodes;
  ?>
  </table>
</tr>
</table>

</body>
</html>

```

Run the website and check that a table of products is displayed, as in the example below. It may be necessary to hold down 'CTRL' whilst clicking the reload icon, to ensure the stylesheet is updated.

Enter order			Plan delivery	Add customer
Stock item	Wholesale price	Quantity required		
AP5613: Alpine jacket	£130.00			
FW6792: Light hiking boot	£32.60			
TN7856: Backpacking tent	£65.00			
CA6782: Sleeping bag	£36.50			
EQ6781: Climbing helmet	£36.90			
WS567: Canadian canoe	£488.30			
MB672: Mountain bike	£163.50			
CN872: Kayak	£398.40			
TR9122: Two-person expedition tent	£89.50			

Return to the **enterOrder.php** file and add the block of program code shown below. This creates input boxes where the quantities of each product required can be entered. The quantity can be selected from a drop-down list of numbers from 0 to 20.

```

echo": ".OutdoorEquipment::$product[$i]->getTitle();
echo"</td>";
$price=OutdoorEquipment::$product[$i]->getPrice();
echo"<td class=stock>£".number_format($price,2) ."</td>";

echo"<td size=10 align='center'>";
echo"<select name='quantity'>";
$quantityWanted=0;
for ($j=0;$j<=20;$j++)
{
    if ($j==$quantityWanted)
        echo"<option selected>".$j;
    else
        echo"<option>".$j;
}
echo"</select></td>";
}
$_SESSION['stockcodes']=$stockcodes;
?>
</table>
</tr>
</table>

```

Save the **enterOrder.php** file and copy it to the server. Run the website and check that drop-down lists are added for each of the product records, as shown below.

Stock item	Wholesale price	Quantity required
AP5613: Alpine jacket	£130.00	0
FW6792: Light hiking boot	£32.60	0
TN7856: Backpacking tent	£65.00	1
CA6782: Sleeping bag	£36.50	2
EQ6781: Climbing helmet	£36.90	3
		4
		5
		6

The customer order page should be able to provide additional details for any of the products, including the photograph image. We will arrange this now. Return to the **enterOrder.php** file and add the lines of program code below. An array **\$display[]** is set up with a value for each product record. A value of 0 will indicate that only the product name should be displayed, whilst 1 will indicate that the text description and photograph should be included.

```

        $_SESSION['login']='YES';
    }
    include('staffMenu.php');
    include('OutdoorEquipment.php');
    $itemcount= OutdoorEquipment::loadStockItems();

    $display=$_SESSION['display'];
    $change=$_REQUEST['changeDisplay'];
    if ($change>0)
    {
        if ($display[$change]==0)
            $display[$change]=1;
        else
            $display[$change]=0;
    }
    $_SESSION['display']=$display;
?>
<p>
<table cellpadding=10>

```

Continue now to add the lines of program code shown below to the **enterOrder.php** file. These create buttons alongside each product record to change the amount of information displayed. When a button is clicked, the `<form>` command causes the page to be reloaded so that the display can be updated.

```

for ($i=1;$i<=$itemcount;$i++)
{
    echo"<form method=post action='enterOrder.php?changeDisplay=". $i ." '>";


    echo"<tr class=stock><td class=stock >";
    $code=OutdoorEquipment::$product[$i]->getStockCode();
    echo $code;
    $stockcodes[$i]=$code;
    echo": ".OutdoorEquipment::$product[$i]->getTitle();

    if ($display[$i]==0)
        echo"<input type=submit style='float:right' value='more...' >";
    else
    {
        echo"<p><table><tr><td>".OutdoorEquipment::$product[$i]->getDescription();
        echo"<td><img src='uploads/'.OutdoorEquipment::$product[$i]->getImage()
        .".' width=200px>";
        echo"<p><input type=submit style='float:right' value='less...' ></table>";
    }
    echo"</form>";

    echo"</td>";
    $price=OutdoorEquipment::$product[$i]->getPrice();
    echo"<td class=stock>£".number_format($price,2) ."</td>";
    echo"<td size=10 align='center'>";
    echo"<select name='quantity'>";

```

Save the **enterOrder.php** file and copy it to the server. Run the website and go to the customer order input page. Click any of the buttons alongside the product records. It should be possible to switch between plain titles and full descriptions of the products.

Stock item	Wholesale price	Quantity required
AP5613: Alpine jacket <input style="float:right" type="button" value="more..."/>	£130.00	0 ▾
FW6792: Light hiking boot <div style="display: flex; align-items: center;"> <div style="flex: 1;"> <p>Made from extremely high quality durable materials. They will provide excellent support and flex for the lifetime of the boot.</p> </div> <div style="flex: 1; text-align: center;">  </div> </div> <input style="float:right; margin-top: 10px;" type="button" value="less..."/>	£32.60	0 ▾
TN7856: Backpacking tent <input style="float:right" type="button" value="more..."/>	£65.00	0 ▾

A problem that you may have noticed is that the order quantities entered in the drop-down boxes are lost when the page is reloaded to change the product display. This can be avoided by storing the quantity required for each product using an array **\$quantity[]**. This array can be stored as a session variable, accessed when the page is reloaded, and the required quantities redisplayed in the input boxes. Add lines of code to the **enterOrder.php** file as shown in the box below.

```

    if ($display[$change]==0)
        $display[$change]=1;
    else
        $display[$change]=0;
}
$_SESSION['display']=$display;

$quantity=$_SESSION['quantity'];
$changeQuantity=$_REQUEST['changeQuantity'];
if ($changeQuantity>0)
{
    $quantityWanted=$_REQUEST['quantity'];
    $quantity[$changeQuantity]=$quantityWanted;
}
$_SESSION['quantity']=$quantity;

?>
<p>
<table cellpadding=10>
<tr><td>
    <table class=stock cellspacing=10px>

```

Now make the changes shown below:

- Insert a **<form>** command at the start of the drop-down quantity selection component.
- Modify the **<select>** command and insert a **\$quantityWanted** variable, replacing two existing lines:


```

                echo"<select name='quantity'>";
                $quantityWanted=0;
            
```

```

if ($display[$i]==0)
    echo"<input type=submit style='float:right' value='more...' >";
else
{
    echo"<p><table><tr><td>".OutdoorEquipment::$product[$i]->getDescription();
    echo"<td><img src='uploads/'.OutdoorEquipment::$product[$i]->getImage()
        .'' width=200px>";
    echo"<p><input type=submit style='float:right' value='less...' ></table>";
}
echo"</form>";
echo"</td>";

echo"<form method=post action='enterOrder.php?changeQuantity=".$i."'>";

$price=OutdoorEquipment::$product[$i]->getPrice();
echo"<td class=stock>£".number_format($price,2);
echo"<td size=10 align='center'>";

echo"<select name='quantity' onchange='this.form.submit()'>";
$quantityWanted=$quantity[$i];

for ($j=0;$j<=20;$j++)
{
    if ($j==$quantityWanted)

```

Insert a **</form>** command at the end of the drop-down quantity selection component code, as shown below.

Continued...

```

Town <input type=text size= '20' id='town'>
</table><br>
<script>
function preload()
{
img1=loadImage("map.png");
}
function setup()
{
createCanvas(1000, 400);
}
function draw()
{
image(img1, 0, 0);
}
</script>
</body>
</html>

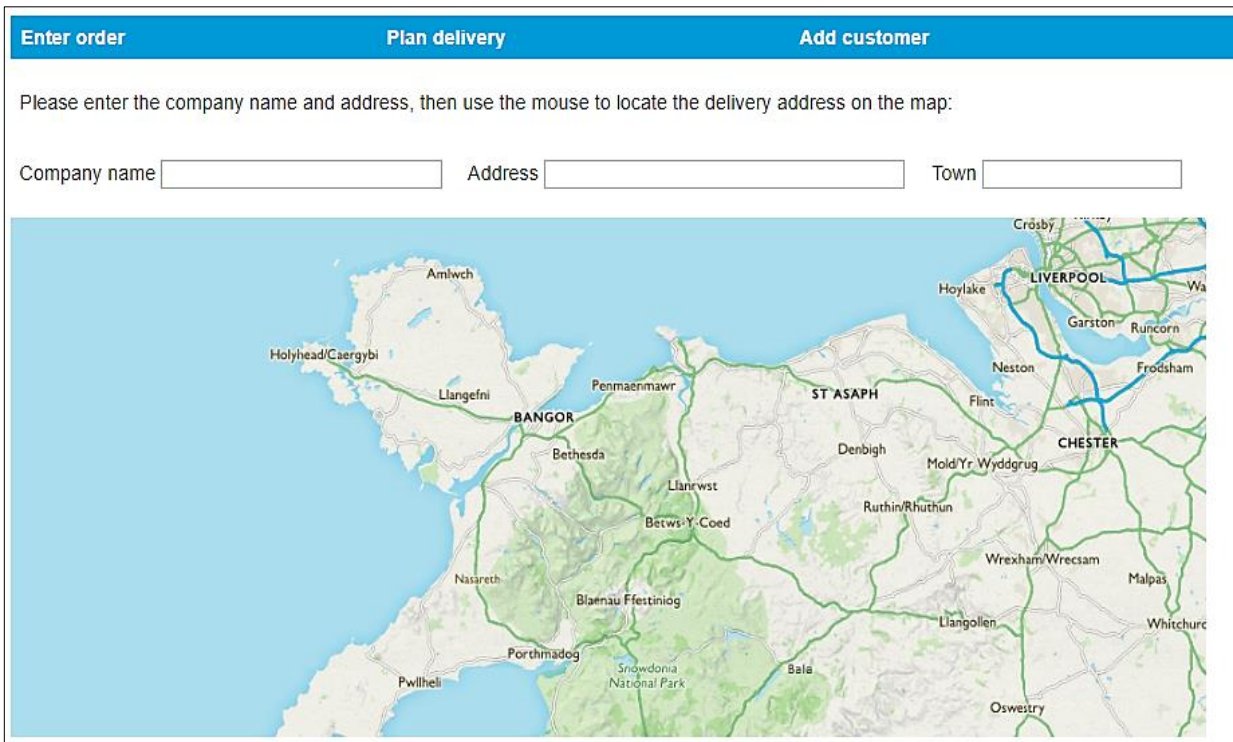
```

Save the file as **addCustomer.php** and copy it to the server.

The page will use the **p5.js** high level extension to JavaScript. Obtain the files **p5.js** and **p5.dom.js** from the developers' website (p5js.org) and copy these to the **delivery** folder on the server.

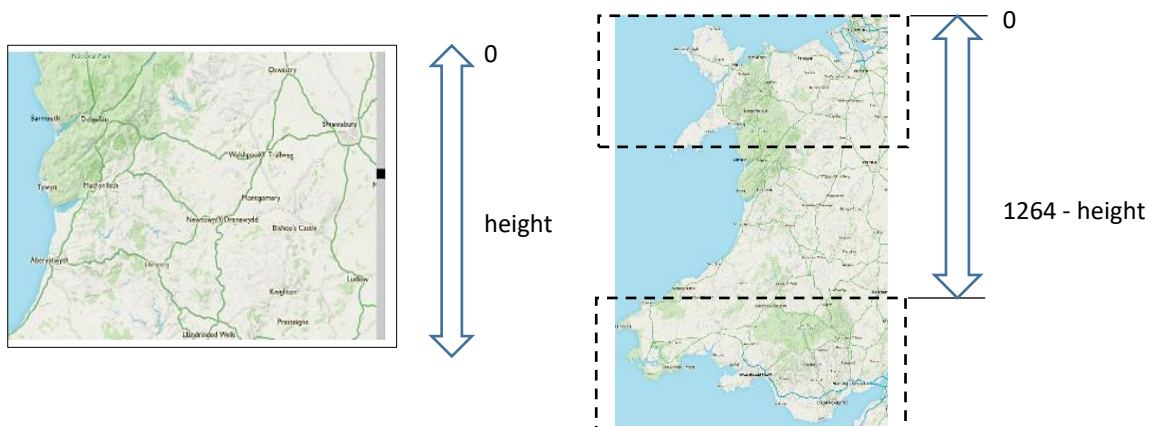
To implement the project for the wholesale company distributing to customers in Wales, a road map of the country should be obtained. This will allow the locations of customers to be recorded. The map used in this example has a width of 1000 pixels and height of 1260 pixels. Save the map image as **map.png** and copy it to the **delivery** folder on the server.

Run the website and select the 'Add customer' option from the menu bar. The page should open, with input boxes for the customer name and address and a section of the map image visible.



The next step is to provide a vertical scroll bar for the map image window. Return to the **addCustomer.php** file and add the lines of program code to the **<script>** block as shown below. A series of JavaScript variables are declared, then commands are added to the **draw()** function.

The **map()** command uses scaling to convert the vertical scroll bar position (between 0 and 400) into the position where the top of the map image should be positioned relative to the top of the display window (between 0 and 864). For example: when the marker is at the bottom of the scroll bar, the map image will be moved upwards by 864 pixels so that only the lowest 400 pixel strip of the map will be displayed in the window.



The position where the map image is drawn can be adjusted by a **translate()** function in p5.js. Add a series of variables at the start of the script block, and code to calculate the correct vertical scroll position for the map image:

```
<script>
  var scrollPosition=0;
  var vPos;
  var ratio;
  var scrollSelected = false;
  var xpos;
  var ypos;
  var show=false;

  function preload()
  {
    img1=loadImage("map.png");
  }

  function setup()
  {
    createCanvas(1000, 400);
  }

  function draw()
  {
    ratio = 1260/400;
    tv = map(scrollPosition, 0, height, 0, 1264-height);
    translate(0, -tv);

    image(img1, 0, 0);
  }
</script>
</body>
```


The final step is to call a function which draws the scroll bar on the right hand side of the map, with the scroll pointer position marked by a darker square.

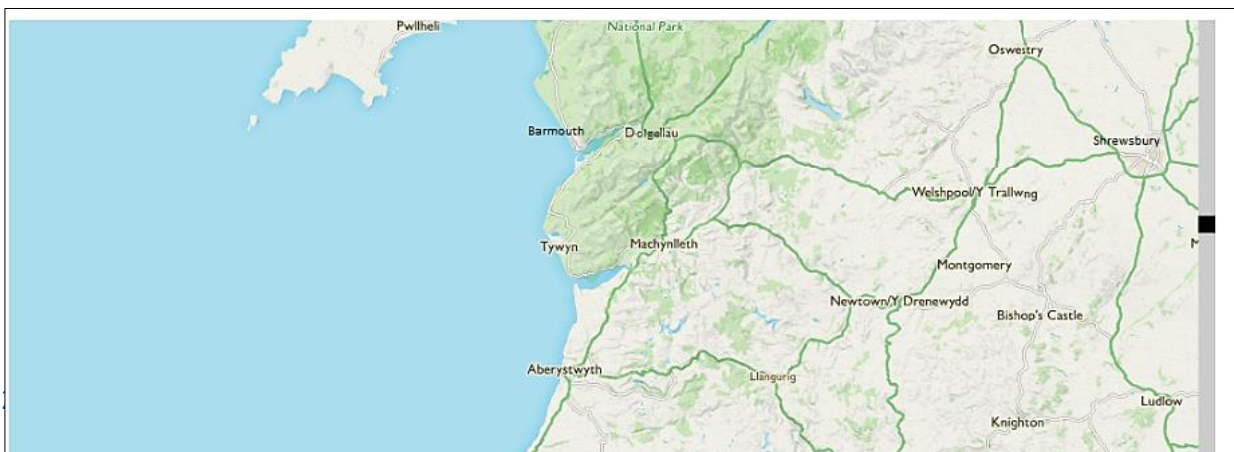
```
image(img1, 0, 0);  
  
scrollbar(scrollPosition);  
x=mouseX;  
y=mouseY;  
if (x>=960)  
{  
  fill(0);  
  rect(986,vPos,14,14);  
  if (mouseIsPressed==true)  
  {  
    scrollPosition=y;  
    if (scrollPosition<0)  
      scrollPosition=0;  
    if (scrollPosition>400)  
      scrollPosition=400;  
  }  
}  
  
</script>  
</body>
```

Insert the **scrollbar ()** function shown below at the end of the **<script>** block.

```
function scrollbar(scrollPosition)  
{  
  noStroke();  
  fill(204);  
  rect(986,0,14,1264);  
  if (scrollSelected==false)  
    fill(102);  
  else  
    fill(40);  
  vPos = scrollPosition * ratio;  
  rect(986,vPos,14,14);  
}  
  
</script>  
</body>  
</html>
```

Save the **addCustomer.php** file and copy it to the server.

Refresh the web page. A scroll bar should now appear to the right of the map. Use the mouse to drag the scroll pointer up and down the scroll bar. The map should move as you do this, allowing the full vertical extent of the map to be displayed.



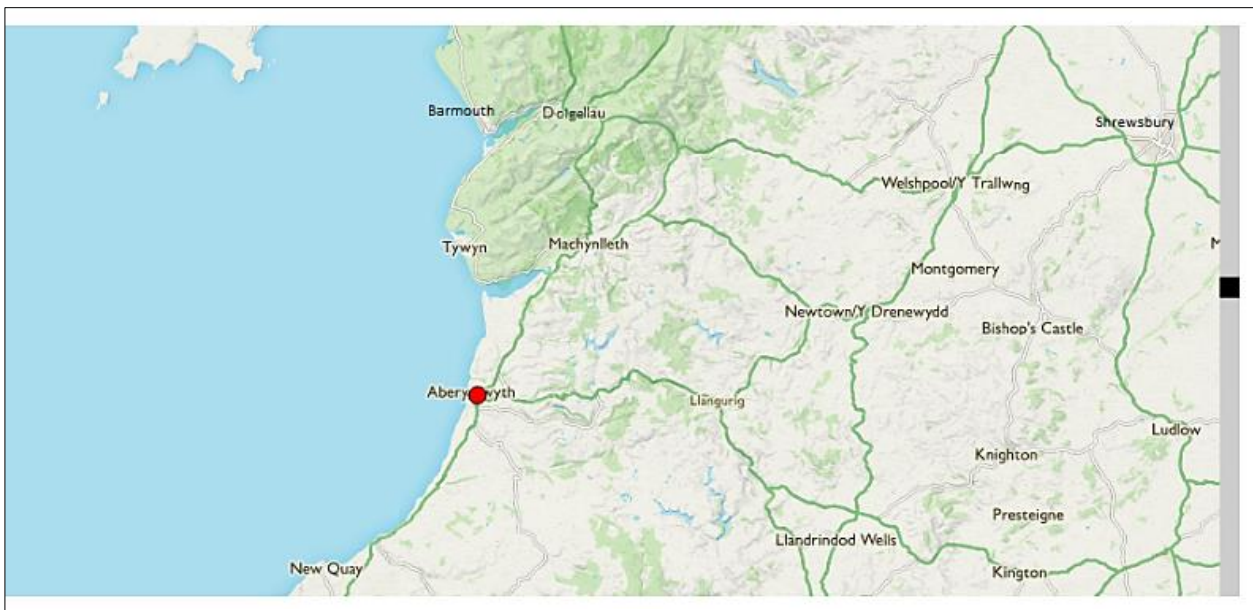
Return to the **addCustomer.php** file and add the lines of program code below. These allow a red circle to be added to the map to indicate the location of the customer.

```

        if (scrollPosition>400)
        {
            scrollPosition=400;
        }
    }
    if (x<960)
    {
        if (mouseIsPressed)
        {
            show=true;
            xpos=x;
            ypos=y+tv;
        }
    }
    if (show==true)
    {
        fill(255,0,0);
        stroke(0);
        ellipse(xpos,ypos,12,12);
    }
}
function scrollbar(scrollPosition)
{
    noStroke();
    fill(204);
    rect(986,0,14,1264);
}

```

Save the **addCustomer.php** file and copy it to the server. Refresh the web page, then click the mouse on the map area. Check that a red circle is plotted, and this remains in position if the map is scrolled.




Return to the **addCustomer.php** file and locate the **setup()** function. Add lines of program code as below. These insert a 'Continue' button below the map window.

```
function setup()
{
  createCanvas(1000, 400);
  button = createButton('Continue');
  button.position(400, 600);
  button.mousePressed(buttonClick);
}
```

Go to the end of the `<script>` block and add a `buttonClick()` function. When the button is pressed, this will collect the customer name, address and map location, then load another web page which will save a record into the database. Save the `addCustomer.php` file and copy it to the server.

```
function buttonClick()
{
  var companyName = document.getElementById("companyName").value;
  var address = document.getElementById("address").value;
  var town = document.getElementById("town").value;
  window.location = "saveCustomer.php?xpos="+int(xpos)+
    "&ypos="+int(ypos)+"&companyName="+companyName+"&address="+
    address+"&town="+town;
}
</script>
</body>
</html>
```

We will now create a table in the database to receive the customer record. Open the PHP MyAdmin website, list the existing tables and select the 'new' option. Create a table with the name '`deliveryCustomer`' and add fields as shown below. The `customerID` field is identified as the primary key, and should be set to auto-increment as records are added to the table.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	customerID 	int(11)			No	None		AUTO_INCREMENT
2	companyName	varchar(40)	latin1_swedish_ci		No	None		
3	address	varchar(30)	latin1_swedish_ci		No	None		
4	town	varchar(30)	latin1_swedish_ci		No	None		
5	x	int(11)			No	None		
6	y	int(11)			No	None		

The next step in saving the customer record is to produce a `DeliveryCustomer` class which will handle file operations. Open a blank file and add the program code below. This defines the attributes for a `DeliveryCustomer` object, which correspond to the fields of the `deliveryCustomer` table.

```
<?
class DeliveryCustomer
{
  public static $customerObj = array();
  private $customerID;
  private $companyName;
  private $address;
  private $town;
  private $x;
  private $y;
}
?>
```

Add a constructor method and a method to save new records into the database.

```
function __construct($customerID,$companyName,$address,$town,$x,$y)
{
    $this->customerID = $customerID;
    $this->companyName = $companyName;
    $this->address = $address;
    $this->town = $town;
    $this->x = $x;
    $this->y = $y;
}

public static function addCustomer($companyName,$address,$town,$x,$y)
{
    include('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="INSERT INTO deliveryCustomer VALUES ('','$companyName',
                                                '$address','$town','$x','$y')";
    $result=mysqli_query($conn, $query);
    mysqli_close($conn);
    echo $query;
}
}
?>
```

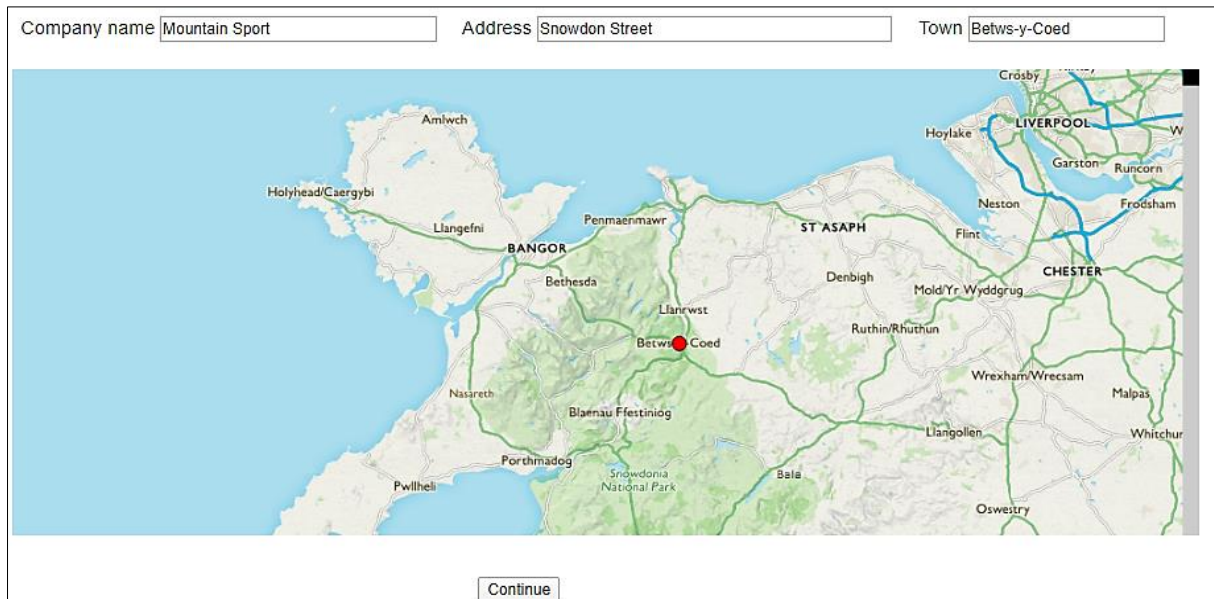
Save the file as **DeliveryCustomer.php** and copy it to the server.

We finally need to create the **saveCustomer** page. This will compile the customer details, then call the method which saves the record into the database table. Open a blank file and add the lines of program code below. Save the file as **saveCustomer.php** and copy it to the server.

```
<?
$companyName=$_REQUEST['companyName'];
$address=$_REQUEST['address'];
$town=$_REQUEST['town'];
$xpos=$_REQUEST['xpos'];
$ypos=$_REQUEST['ypos'];
$companyName=str_replace("'", "`", $companyName);
$address=str_replace("'", "`", $address);
include('DeliveryCustomer.php');
DeliveryCustomer::addCustomer($companyName,$address,$town,$xpos,$ypos);
header('Location: enterOrder.php');
?>
```

Notice that lines of program have been added to convert any apostrophe characters (') into back-tick characters (`). This avoids any problems when the data is saved into the database table, as the database software uses apostrophes as markers for the ends of fields.

We now have all the resources in place to create and store a customer record. Return to the **enterOrder** web page. Add a company name, address and town, then locate the customer on the map as in the example below.



Click the 'Continue' button. The program should return to the enterOrder page. Go to the database, open the **deliveryCustomer** table and check that a correct record has been inserted. If all is well, repeat the procedure to add a series of customers in different towns around Wales.

We can now complete the customer selection function on the enterOrder page. Begin by re-opening the **DeliveryCustomer.php** class file. Add a **loadCustomers()** method as shown below.

```

public static function loadCustomers()
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM deliveryCustomer ORDER BY companyName";
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    for ($i=1;$i<=$num; $i++)
    {
        $row=mysqli_fetch_assoc($result);
        $customerID=$row["customerID"];
        $companyName=$row["companyName"];
        $address=$row["address"];
        $town=$row["town"];
        $x=$row["x"];
        $y=$row["y"];
        $obj = new DeliveryCustomer($customerID,$companyName,
                                   $address,$town,$x,$y);
        DeliveryCustomer::$customerObj[$i] = $obj;
    }
    return $num;
}
}
?>

```

Also add a series of **get()** methods as shown below, which will allow the program to obtain data values from the customer objects.

```

    public function getCustomerID(){return $this->customerID;}
    public function getCompanyName(){return $this->companyName;}
    public function getAddress(){return $this->address;}
    public function getTown(){return $this->town;}
    public function getX(){return $this->x;}
    public function getY(){return $this->y;}
}
?>

```

Save the **DeliveryCustomer.php** file and copy it to the server. Open the **enterOrder.php** file and add the block of program code shown below.

```

$_SESSION['quantity']=$quantity;
?>
<p>
<table cellpadding=10>


<style> select {font-size: 16px;} </style>
<tr><td>
<?
    include('DeliveryCustomer.php');
    $count = DeliveryCustomer::loadCustomers();
    $customerWanted=$_REQUEST['customerID'];
    if ($customerWanted>0)
        $_SESSION['customerID']=$customerWanted;
    $customerWanted=$_SESSION['customerID'];
    echo"<form method=post action='enterOrder.php'>";
    echo"<table cellpadding=2><tr><td>Customer</td>";
    echo"<td><select name='customerID' onchange='this.form.submit()'>";
    echo "<option value=0>";
    $address='&nbsp;';
    $town='&nbsp;';
    for ($i=1;$i<=$count;$i++)
    {
        $customerID=DeliveryCustomer::$customerObj[$i]->getCustomerID();
        $companyName=DeliveryCustomer::$customerObj[$i]->getCompanyName();
        if($customerWanted==$customerID)
        {
            echo "<option selected value=".$customerID.">".$companyName;
            $address=DeliveryCustomer::$customerObj[$i]->getAddress();
            $town=DeliveryCustomer::$customerObj[$i]->getTown();
        }
        else
        {
            echo "<option value=".$customerID.">".$companyName;
        }
    }
    echo"</select></td>";
    echo"<tr><td></td><td>".$address."</td></tr>";
    echo"<tr><td></td><td>".$town."</td></tr>";
    echo"</table>";
    echo"</form>";
?>

<tr><td>
<table class=stock cellspacing=10px>

```

This block of code begins by loading all the customer records from the database, then sets up a selection box in which the company names will be displayed. When a company is selected, the corresponding **customerID** is stored, the page is reloaded and the companyID is retrieved. This companyID value is then used to display the correct company name in the selection box window, and to add the address and town of the customer in a small table underneath.

Save the **enterOrder.php** file and copy it to the server. Run the **enterOrder** page, check that customers can be selected and the corresponding address and town is then displayed.

Enter order		Plan delivery	Add customer
Customer: <input type="text" value="Swansea Leisure"/> Station Road Swansea			
Stock item		Wholesale price	Quantity required
AP5613: Alpine jacket <input style="float: right;" type="button" value="more..."/>		£130.00	<input type="text" value="0"/>
FW6792: Light hiking boot Made from extremely high quality durable materials. They will provide excellent support and flex for the lifetime of the boot.		£32.60	<input type="text" value="6"/>
			

We will now prepare to save orders into the database. Two new tables will be needed. Open the PHP MyAdmin website. Create an **outdoorEquipmentOrder** table and add fields as shown below. The **orderID** field is identified as the primary key, and should be set to auto-increment as records are added to the table.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	orderID	int(11)			No	None		AUTO_INCREMENT
2	customerID	int(11)			No	None		
3	orderDate	varchar(10)	latin1_swedish_ci		No	None		
4	delivered	varchar(10)	latin1_swedish_ci		No	None		
5	deliveryDate	varchar(12)	latin1_swedish_ci		No	None		

Also create an **equipmentOrderItem** table. The **orderItemID** field is identified as the primary key, and is set to auto-increment.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	orderItemID	int(11)			No	None		AUTO_INCREMENT
2	orderID	int(11)			No	None		
3	stockcode	varchar(10)	latin1_swedish_ci		No	None		
4	quantity	int(11)			No	None		

We will create an **OutdoorEquipmentOrder** class to handle the transfer of records to the database. Open a blank file and add the program code shown below.

```

<?
class OutdoorEquipmentOrder
{
    public static $orders = array();
    private $orderID;
    private $customerID;
    private $orderDate;
    private $delivered;
    private $deliveryDate;

    function __construct($orderID,$customerID,$orderDate,$delivered,$deliveryDate)
    {
        $this->orderID = $orderID;
        $this->customerID = $customerID;
        $this->orderDate = $orderDate;
        $this->delivered = $delivered;
        $this->deliveryDate = $deliveryDate;
    }

    public static function addOrder($customerID,$orderDate,$delivered,$deliveryDate)
    {
        include('user.inc');
        $conn = new mysqli(localhost, $username, $password, $database);
        if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
        $query="INSERT INTO outdoorEquipmentOrder VALUES ('','$customerID',
            '$orderDate','$delivered','$deliveryDate')";
        echo"<p>".$query;
        $result=mysqli_query($conn, $query);
        $newOrderID = mysqli_insert_id($conn);
        mysqli_close($conn);
        return $newOrderID;
    }
}
?>

```

Save the file as **OutdoorEquipmentOrder.php** and copy it to the server. The **addOrder()** method saves the record into the `outdoorEquipmentOrder` table. The computer creates an `orderID` as an auto-number. This value is then returned as the variable **\$newOrderID**, which can be used to link to the individual item records making up the order.

We can now create an **EquipmentOrderItem** class in a similar way. Open a blank file and add the attributes shown.

```

<?
class EquipmentOrderItem
{
    public static $items = array();
    private $orderItemID;
    private $orderID;
    private $stockcode;
    private $quantity;
}
?>

```

Insert a constructor method and a method to add order items to the database table, as shown below.


```

function __construct($orderItemID,$orderID,$stockcode,$quantity)
{
    $this->orderItemID = $$orderItemID;
    $this->orderID = $orderID;
    $this->stockcode = $stockcode;
    $this->quantity = $quantity;
}

public static function addItem($orderID,$stockcode,$quantity)
{
    include('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="INSERT INTO equipmentOrderItem VALUES ('','$orderID',
                                                    '$stockcode','$quantity')";

    echo"<p>".$query;
    $result=mysqli_query($conn, $query);
    mysqli_close($conn);
}
}
?>

```

Save the file as **EquipmentOrderItem.php** and copy it to the server.

Return to the **enterOrder.php** page. Add the lines of code shown below. These create an 'Enter order' button. When clicked, this will load another web page where the order data will be saved to the database. Save **enterOrder.php** and copy it to the server.

```

echo"<tr><td></td><td>".$address."</td></tr>";
echo"<tr><td></td><td>".$town."</td></tr>";
echo"</table>";
echo"</form>";
?>

<form method=post action='saveOrder.php'>
<tr><td><center><input type=submit value='Enter order'></center></td></tr>
</form>

<tr><td>
    <table class=stock cellpadding=10px>
        <col width="800">

```

We will now create a **saveOrder** page which will handle the upload of data.

The program begins by collecting the **customerID**, and generating the current date which will be used as the **order date**. This data is then saved as a record in the **outdoorEquipmentOrder** table. The new order number is returned as a variable.

The next step is to collect the arrays containing the **stock codes** and **quantities** required for each of the products. These arrays had been stored as session variables by the enterOrder web page.

Finally, a loop operates for each of the products. If one or more of the current product has been ordered, a record is saved into the **equipmentOrderItem** table. This record contains the **stock code, quantity** required, and the **orderID** for the overall order.

Open a blank file and add the lines of program code below. Save the file as **saveOrder.php** and copy it to the server.


```

<?
    session_start();
    $customerWanted=$_SESSION['customerID'];
    $today = date("Y-m-d");
    include('OutdoorEquipmentOrder.php');
    include('EquipmentOrderItem.php');
    $newOrderID=OutdoorEquipmentOrder::addOrder($customerWanted,$today,'','');
    $stockcodes=$_SESSION['stockcodes'];
    $count=count($stockcodes);
    $quantity=$_SESSION['quantity'];
    for ($i=1;$i<=$count;$i++)
    {
        if ($quantity[$i]>0)
        {
            EquipmentOrderItem::addItem($newOrderID,$stockcodes[$i],$quantity[$i]);
        }
    }
    $_SESSION['quantity']='';
    header('Location: deliveries.php');
?>

```

Before testing the program, save a blank file as **deliveries.php** and copy it to the server. Run the website. Go to the **'Enter order'** page and make an order, selecting a customer and specifying the quantities required for several products. Click the **'Enter order'** button. The program will save the order in the database and then load the blank **deliveries.php** page.

Open the PHP MySQL website and carefully check that the order has been saved correctly:

- Open the **outdoorEquipmentOrder** table. An order record should be present, with the customerID shown as an integer.
- Go to the **deliveryCustomer** table and check that the correct customer ID was used.
- Return to the **outdoorEquipmentOrder** table and make a note of the orderID which was allocated as an auto-number.
- Go now to the **equipmentOrderItem** table. Check that a record has been created for each of the products ordered, and that the stock codes and quantities are shown correctly. Also check that the orderID matches with the value given in the **outdoorEquipmentOrder** table.

We can now move on to create a table of the customer orders. Re-open the blank **deliveries.php** file and add the lines of program code in the two boxes below.

```

<html>
<head>
    <title> Delivery route </title>
    <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
<?
include('staffMenu.php');
include('OutdoorEquipment.php');
include('OutdoorEquipmentOrder.php');
include('DeliveryCustomer.php');
include('EquipmentOrderItem.php');
?>
<p>
    <table class=stock cellspacing=10px>
        <tr class=stock>
            <th class=stock>OrderID</th>

```

```

        <th class=stock>Order date</th>
        <th class=stock>Customer</th>
        <th class=stock>Town</th>
        <th class=stock></th>
        <th class=stock>Delivered</th>
        <th class=stock>Select for delivery</th>
        <th class=stock>Delivery date</th>
        <th class=stock>Delivery completed</th>
        <th class=stock></th>
    </tr>
</table>
</body>
</html>

```

Save the **deliveries.php** file and copy it to the server. Run the website and go to the 'Plan delivery' menu option. The menu bar and table headings should be displayed.



We must now add some methods to the object classes to provide data for the table of orders. Open the **OutdoorEquipmentOrder.php** file and add the methods shown below. The **loadOrders()** method will obtain all order records from the database and create a set of objects. The **get()** methods will allow the individual attributes of these objects to be displayed on the web page.

```

public static function loadOrders()
{
    include('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM outdoorEquipmentOrder";
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    for ($i=1;$i<=$num; $i++)
    {
        $row=mysqli_fetch_assoc($result);
        $orderID=$row["orderID"];
        $customerID=$row["customerID"];
        $orderDate=$row["orderDate"];
        $delivered=$row["delivered"];
        $deliveryDate=$row["deliveryDate"];
        $obj = new OutdoorEquipmentOrder($orderID, $customerID,
                                         $orderDate,$delivered,$deliveryDate);
        OutdoorEquipmentOrder::$orders[$i] = $obj;
    }
    return $num;
}

public function getOrderID(){return $this->orderID;}
public function getCustomerID(){return $this->customerID;}
public function getOrderDate(){return $this->orderDate;}
public function getDelivered(){return $this->delivered;}
public function getDeliveryDate(){return $this->deliveryDate;}
}
?>

```

Save the **OutdoorEquipmentOrder.php** file and copy it to the server.

Open the **DeliveryCustomer.php** class file and add the **loadCustomerByID()** method shown below. This will take the ID number of a customer, obtained from an order record, and use it to select the corresponding customer record in the database. This will then be converted to an object, so that the individual attributes of name, address and town can be obtained for display on the web page. Save the **DeliveryCustomer.php** file and copy it to the server.

```

public static function loadCustomerByID($IDwanted)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM deliveryCustomer WHERE customerID = ".$IDwanted;
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    $row=mysqli_fetch_assoc($result);
    $customerID=$row["customerID"];
    $companyName=$row["companyName"];
    $address=$row["address"];
    $town=$row["town"];
    $x=$row["x"];
    $y=$row["y"];
    $obj = new DeliveryCustomer($customerID,$companyName,$address,$town,$x,$y);
    DeliveryCustomer::$customerObj[1] = $obj;
}
}
?>

```

Return now to the **deliveries.php** file and add the block of program code shown below.

```

<th class=stock>Delivery completed</th>
<th class=stock></th>
</tr>
<?
$orderCount=OutdoorEquipmentOrder::loadOrders();
for ($i=1; $i<=$orderCount;$i++)
{
    $delivered = OutdoorEquipmentOrder::$orders[$i]->getDelivered();
    $orderID = OutdoorEquipmentOrder::$orders[$i]->getOrderID();
    $orderDate = OutdoorEquipmentOrder::$orders[$i]->getOrderDate();
    $IDwanted = OutdoorEquipmentOrder::$orders[$i]->getCustomerID();
    $rearrangedDate = substr($orderDate,8,2)."-".substr($orderDate,5,2).
        "-".substr($orderDate,0,4);
    echo"<tr><td class=stock>".$orderID."</td>";
    echo"<td class=stock>".$rearrangedDate."</td>";
    DeliveryCustomer::loadCustomerByID($IDwanted);
    $companyName = DeliveryCustomer::$customerObj[1]->getCompanyName();
    $town = DeliveryCustomer::$customerObj[1]->getTown();
    echo"<td class=stock>". $companyName."</td>";
    echo"<td class=stock>". $town."</td>";
    echo"<td class=stock><input type=submit value='details'></td>";
    echo"<td class=stock>". $delivered."</td>";
    echo"</tr>";
}
?>
</table>
</body>

```

Save the **deliveries.php** file and copy it to the server. Refresh the 'Plan delivery' web page. The program code listed above contains a loop which operates for each order. The order ID and order date are obtained from the **OutdoorEquipmentOrder** object. The customerID is also obtained, then this is used to select the corresponding **DeliveryCustomer** object, which provides the company name and town.

Enter order		Plan delivery			Add customer		Add stock item	
OrderID	Order date	Customer	Town		Delivered	Select for delivery	Delivery date	Delivery completed
1	06-10-2019	Snowdonia Mountainsport	Betws-y-Coed	<input type="button" value="details"/>				
2	06-10-2019	Bay Watersports	Cardiff	<input type="button" value="details"/>				
4	06-10-2019	Mountain Gear	Newtown	<input type="button" value="details"/>				
6	06-10-2019	Cader Idris Climbing Shop	Dolgellau	<input type="button" value="details"/>				
7	06-10-2019	Bay Watersports	Cardiff	<input type="button" value="details"/>				
8	06-10-2019	Pwllheli Surf Shop	Pwllheli	<input type="button" value="details"/>				
9	06-10-2019	Lakeside Watersports	Bala	<input type="button" value="details"/>				

A button has been added to each record which will allow full details of the order to be displayed, including the quantities of each product required. Finally, a column indicates when an order has been delivered to the customer.

We will now develop the record keeping system for deliveries. Return to the **deliveries.php** file and add the program code shown below. Save the **deliveries.php** file and copy it to the server, then refresh the web page.

```

echo"<td class=stock>". $town."</td>";
echo"<td class=stock><input type=submit value='details'></td>";
echo"<td class=stock>". $delivered."</td>";

if ($delivered=='YES')
{
    echo"<td class=stock></td>";
    $deliveredDate = OutdoorEquipmentOrder::$orders[$i]->getDeliveryDate();
    echo"<td class=stock>". $deliveredDate."</td>";
}
else
{
    echo"<td class=stock align='center'><input type=checkbox id='$orderID'
        name='select' onclick='checkboxUpdate()'></td>";
}
if ($delivered=='YES')
{
    echo"<td class=stock></td><td class=stock></td>";
}
else
{
    echo"<form method=post action=deliveries.php?orderID=".$orderID.
        "&delivery=YES>";
    echo"<td class=stock><input type=date name=deliveryDate value='".
        $dateDelivered."></td>";
    echo"<td class=stock align='center'><input type=checkbox
        name='deliverCheckbox'></td>";
    echo"<td ><input type=submit value='update'></td>";
    echo"</form>";
}

echo"</tr>";
}
?>
</table>

```

Checkboxes have been added to allow particular orders to be included in a delivery journey. A button allows an order to be marked as 'delivery completed' and the delivery date recorded.

OrderID	Order date	Customer	Town		Delivered	Select for delivery	Delivery date	Delivery completed	
1	06-10-2019	Snowdonia Mountainsport	Betws-y-Coed	<input type="button" value="details"/>		<input type="checkbox"/>	<input type="text" value="dd/mm/yyyy"/>	<input type="checkbox"/>	<input type="button" value="update"/>
2	06-10-2019	Bay Watersports	Cardiff	<input type="button" value="details"/>		<input type="checkbox"/>	<input type="text" value="dd/mm/yyyy"/>	<input type="checkbox"/>	<input type="button" value="update"/>
4	06-10-2019	Mountain Gear	Newtown	<input type="button" value="details"/>	YES		2019-11-04		
6	06-10-2019	Cader Idris Climbing Shop	Dolgellau	<input type="button" value="details"/>	YES		2019-10-16		
7	06-10-2019	Bay Watersports	Cardiff	<input type="button" value="details"/>	YES		2019-10-08		
8	06-10-2019	Pwllheli Surf Shop	Pwllheli	<input type="button" value="details"/>		<input type="checkbox"/>	<input type="text" value="dd/mm/yyyy"/>	<input type="checkbox"/>	<input type="button" value="update"/>
9	06-10-2019	Lakeside Watersports	Bala	<input type="button" value="details"/>		<input type="checkbox"/>	<input type="text" value="dd/mm/yyyy"/>	<input type="checkbox"/>	<input type="button" value="update"/>

The next feature which we can add to the 'Plan delivery' web page is an option to list **all orders**, or only those **awaiting delivery**. This will be selected by means of radio buttons:

Enter order		Plan delivery			Add customer	
<input type="radio"/> Display all orders <input checked="" type="radio"/> Display orders awaiting delivery						
OrderID	Order date	Customer	Town		Delivered	Select for delivery

Return to the **deliveries.php** file and add the lines of program code show below.

```

include('OutdoorEquipmentOrder.php');
include('DeliveryCustomer.php');
include('EquipmentOrderItem.php');
?>
<p>
<form method=post action=deliveries.php>
<?
  $display=$_REQUEST['display'];
  if ($display=='all')
  {
    echo"<input type='radio' name='display' value='all'
      onclick='this.form.submit()' checked> Display all orders";
    echo"<input type='radio' name='display' value='awaiting'
      onclick='this.form.submit()> Display orders awaiting delivery";
  }
  else
  {
    echo"<input type='radio' name='display' value='all'
      onclick='this.form.submit()> Display all orders";
    echo"<input type='radio' name='display' value='awaiting'
      onclick='this.form.submit()' checked> Display orders awaiting delivery";
  }
?>
</form>
<p>
<table class=stock cellpadding=10px>
<tr class=stock>
  <th class=stock>OrderID</th>
  <th class=stock>Order date</th>

```

Continuing to work on the **deliveries.php** file, add the line of code shown below.

```

        <th class=stock>Delivery completed</th>
        <th class=stock></th>
    </tr>
<?
$orderCount=OutdoorEquipmentOrder::loadOrders();
for ($i=1; $i<=$orderCount;$i++)
{
    $delivered = OutdoorEquipmentOrder::$orders[$i]->getDelivered();
    if (($display=='all')||($delivered !="YES"))
    {
        $orderId = OutdoorEquipmentOrder::$orders[$i]->getOrderID();
        $orderDate = OutdoorEquipmentOrder::$orders[$i]->getOrderDate();
        $IDwanted = OutdoorEquipmentOrder::$orders[$i]->getCustomerID();
    }
}

```

Also add a closing bracket after the end of the table row.

```

        echo"<td ><input type=submit value='update'></td>";
        echo"</form>";
    }
    echo"</tr>";
}
?>
</table>
</body>

```

Save the **deliveries.php** file, copy it to the server and refresh the web page. When this page is completed, it will be possible to display either all orders or just those waiting for delivery.

Display all orders
 Display orders awaiting delivery

OrderID	Order date	Customer	Town		Delivered	Select for delivery	Delivery date
1	06-10-2019	Snowdonia Mountainsport	Betws-y-Coed	details		<input type="checkbox"/>	dd/mm/yyyy
2	06-10-2019	Bay Watersports	Cardiff	details		<input type="checkbox"/>	dd/mm/yyyy
8	06-10-2019	Pwllheli Surf Shop	Pwllheli	details		<input type="checkbox"/>	dd/mm/yyyy

Display all orders
 Display orders awaiting delivery

OrderID	Order date	Customer	Town		Delivered	Select for delivery	Delivery date
1	06-10-2019	Snowdonia Mountainsport	Betws-y-Coed	details		<input type="checkbox"/>	dd/mm/yyyy
2	06-10-2019	Bay Watersports	Cardiff	details		<input type="checkbox"/>	dd/mm/yyyy
4	06-10-2019	Mountain Gear	Newtown	details	YES		2019-11-04
6	06-10-2019	Cader Idris Climbing Shop	Dolgellau	details	YES		2019-10-16
7	06-10-2019	Bay Watersports	Cardiff	details	YES		2019-10-08
8	06-10-2019	Pwllheli Surf Shop	Pwllheli	details		<input type="checkbox"/>	dd/mm/yyyy

We can now arrange for the details of orders to be displayed when required. This will be done by clicking the 'details' buttons.

Return to the **deliveries.php** file and add a block of code at the beginning of the program. This accesses an array **\$details[]** which has an entry for each of the orders. The value is set to 0 if only the order summary is required, and 1 if a full list of the items ordered are to be displayed.

```
<?
    session_start();
    $details=$_SESSION['details'];
    $changeDetails=$_REQUEST['changeDetails'];
    if ($details[$changeDetails]==0)
        $details[$changeDetails]=1;
    else
        if ($details[$changeDetails]==1)
            $details[$changeDetails]=0;
    $_SESSION['details']=$details;
?>

<html>
<head>
    <title> Delivery route </title>
    <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
```

Go now to the **<table>** section of the program where the 'details' button is displayed. Create a **<form>** block around the input line. This will cause the page to be reloaded when the button is pressed, with the orderID included in the URL. The entry for this order in the **\$details[]** array will then be updated.

```
$town = DeliveryCustomer::$customerObj[1]->getTown();
echo"<td class=stock>". $companyName."</td>";
echo"<td class=stock>". $town."</td>";

echo"<form method=post action='deliveries.php?display=".
    $display."&changeDetails=".$i.">";

echo"<td class=stock><input type=submit value='details'></td>";

echo"</form>";

echo"<td class=stock>". $delivered."</td>";
if ($delivered=='YES')
{
    echo"<td class=stock></td>";
```

Go to a point near the end of the **<table>** and insert the block of code shown below.

```
    echo"<td ><input type=submit value='update'>";
    echo"</form>";
}
echo"</tr>";

if ($details[$i]==1)
{
    echo"<tr><td>Details</td></tr>";
}
}
}
?>
</table>
</body>
```

The program checks the `$details[]` array as each order is displayed in the table. If a value of 1 is found for the current line, a temporary test message with the word 'Details' will be displayed. This will allow us to test that the program is working correctly.

Save the `deliveries.php` file and copy it to the server.

Run the 'Plan delivery' web page. Click on any of the 'details' buttons and a message line should appear underneath. Clicking a second time should cancel the message line.

2	06-10-2019	Bay Watersports	Cardiff	details	<input type="checkbox"/>	dd/mm/yyyy
8	06-10-2019	Pwllheli Surf Shop	Pwllheli	details	<input type="checkbox"/>	dd/mm/yyyy
Details						
9	06-10-2019	Lakeside Watersports	Bala	details	<input type="checkbox"/>	dd/mm/yyyy
10	07-10-2019	Cader Idris Climbing Shop	Dolgellau	details	<input type="checkbox"/>	dd/mm/yyyy

Return to the `deliveries.php` file. We can now produce a list of items ordered. Replace the "echo" test message with the lines of program code shown below. These will display the stock code and quantity of each product included in the order. Save the file and copy it to the server.

```

        echo"<td ><input type=submit value='update'>";
        echo"</form>";
    }
    echo"</tr>";
    if ($details[$i]==1)
    {
        $itemCount=EquipmentOrderItem::loadItemsByOrderID($orderID);
        for ($n=1;$n<=$itemCount;$n++)
        {
            $stockcode=EquipmentOrderItem::$items[$n]->getStockcode();
            $quantity=EquipmentOrderItem::$items[$n]->getQuantity();
            echo"<tr><td>".$stockcode."</td>";
            echo"<td>".$quantity."</td>";
            echo"</tr>";
        }
    }
}

```

Before running the program, we will need to add methods to `EquipmentOrderItem` class to select the required order from the database and display the attributes on the page.

Open the `EquipmentOrderItem.php` file and insert a set of `get()` methods to allow access to the object attributes:

```

    public function getOrderItemID(){return $this->orderItemID;}
    public function getOrderID(){return $this->orderID;}
    public function getStockcode(){return $this->stockcode;}
    public function getQuantity(){return $this->quantity;}
}
?>

```

Continuing to work in the `EquipmentOrderItem.php` file, add a `loadItemsByOrderID()` method as shown below.


```

public static function loadItemsByOrderID($IDwanted)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM equipmentOrderItem WHERE orderID =".$IDwanted;
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    $i=1;
    while ($i <= $num)
    {
        $row=mysqli_fetch_assoc($result);
        $orderItemID=$row["orderItemID"];
        $orderID=$row["orderID"];
        $stockcode=$row["stockcode"];
        $quantity=$row["quantity"];
        $obj = new EquipmentOrderItem($orderItemID,$orderID,
                                     $stockcode,$quantity);
        EquipmentOrderItem::$items[$i] = $obj;
        $i++;
    }
    return $num;
}
}
?>

```

Save **EquipmentOrderItem.php** and copy it to the server.

Refresh the 'Plan delivery' web page and click any of the 'details' buttons. A list of the stock codes and quantities ordered should now be shown.

OrderID	Order date	Customer	Town		Delivered
1	06-10-2019	Snowdonia Mountainsport	Betws-y-Coed	<input type="button" value="details"/>	
FW6792 4 AP5613 2 WS567 1 TN7856 1					
2	06-10-2019	Bay Watersports	Cardiff	<input type="button" value="details"/>	
CN872 2					
4	06-10-2019	Mountain Gear	Newtown	<input type="button" value="details"/>	
6	06-10-2019	Cader Idris Climbing Shop	Dolgellau	<input type="button" value="details"/>	
AP5613 2 CA6782 1					
7	06-10-2019	Bay Watersports	Cardiff	<input type="button" value="details"/>	

We can now add the product title and price for each of the items. To do this, an additional method will be required in the **OutdoorEquipment** class.

Open the file **OutdoorEquipment.php** and add the **loadByStockCode()** method shown below. Save the **OutdoorEquipment.php** file and copy it to the server.

```

public static function loadByStockCode($codeWanted)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM outdoorEquipment WHERE stockcode='".$codeWanted.'";
    $result=mysqli_query($conn, $query);
    mysqli_close($conn);
    $row=mysqli_fetch_assoc($result);
    $productID=$row["productID"];
    $stockCode=$row["stockcode"];
    $title=$row["title"];
    $category=$row["category"];
    $description=$row["description"];
    $picture=$row["picture"];
    $price=$row["price"];
    $obj = new OutdoorEquipment($productID, $stockCode, $category,
                                $title,$description, $picture, $price);
    OutdoorEquipment::$product[0] = $obj;
}
}
?>

```

Return to the **deliveries.php** file and add the lines of program code shown below. These will display the title and price for each product, calculate each item total and the total cost for the order.

```

if ($details[$i]==1)
{
    $itemCount=EquipmentOrderItem::loadItemsByOrderID($orderId);
    $total=0;
    for ($n=1;$n<=$itemCount;$n++)
    {
        $stockcode=EquipmentOrderItem::$items[$n]->getStockcode();
        $quantity=EquipmentOrderItem::$items[$n]->getQuantity();
        echo"<tr><td>".$stockcode."</td>";
        echo"<td>".$quantity."</td>";

        OutdoorEquipment::loadByStockCode($stockcode);
        $productTitle=OutdoorEquipment::$product[0]->getTitle();
        $price=OutdoorEquipment::$product[0]->getPrice();
        echo"<td>".$productTitle."</td>";
        echo"<td>@ £".number_format($price,2)."</td>";
        $itemtotal=$price*$quantity;
        echo"<td><td>£".number_format($itemtotal,2)."</td>";
        $total=$total+$itemtotal;
        echo"<td colspan=4></td>";

        echo"</tr>";
    }

    echo"<tr><td colspan=7></td><td>Total:</td>";
    echo"<td>£".number_format($total,2)."</td>";
    echo"<td></td></tr>";
}
}
?>
</table>

```

Save **deliveries.php** and copy it to the server. Run the website and go to the 'Plan delivery' page. Click any of the 'details' buttons. Full details of the order should now be displayed, as in the example below.

OrderID	Order date	Customer	Town		Delivered	Select for delivery	Delivery date	Delivery completed
1	06-10-2019	Snowdonia Mountainsport	Betws-y-Coed	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>
FW6792	4	Light hiking boot	@ £32.60		£130.40			
AP5613	2	Alpine jacket	@ £130.00		£260.00			
WS567	1	Canadian canoe	@ £488.30		£488.30			
TN7856	1	Backpacking tent	@ £65.00		£65.00			
Total:								£943.70
2	06-10-2019	Bay Watersports	Cardiff	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>
4	06-10-2019	Mountain Gear	Newtown	details	YES		2019-11-04	

The next task on the 'Plan delivery' page is to handle the recording of deliveries. When a delivery is made, the date will be entered, the 'Delivery completed' checkbox will be ticked, and the 'update' button clicked. The delivery record in the database will then be updated.

OrderID	Order date	Customer	Town		Delivered	Select for delivery	Delivery date	Delivery completed	
1	06-10-2019	Snowdonia Mountainsport	Betws-y-Coed	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>	update
2	06-10-2019	Bay Watersports	Cardiff	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>	update
8	06-10-2019	Pwllheli Surf Shop	Pwllheli	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>	update
9	06-10-2019	Lakeside Watersports	Bala	details		<input type="checkbox"/>	dd/mm/yyyy	<input checked="" type="checkbox"/>	update
10	07-10-2019	Cader Idris Climbing Shop	Dolgellau	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>	update
13	28-10-2019	Sailing Centre	New Quay	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>	update
14	31-10-2019	Swansea Leisure	Swansea	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>	update
33	14-03-2020	Lakeside Watersports	Bala	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>	update
34	14-03-2020	Barry Island Watersports	Barry	details		<input type="checkbox"/>	dd/mm/yyyy	<input type="checkbox"/>	update

Re-open the **deliveries.php** file and add the block of program code shown below. This is called when an 'update' button is clicked. The delivery date is obtained from the input box, the database record is updated, then the page is refreshed.

```
include('OutdoorEquipmentOrder.php');
include('DeliveryCustomer.php');
include('EquipmentOrderItem.php');

$orderCount=OutdoorEquipmentOrder::loadOrders();
$delivery=$_REQUEST['delivery'];
if ($delivery=='YES')
{
    $orderIDwanted=$_REQUEST['orderID'];
    $deliveryCheckbox=$_REQUEST['deliverCheckbox'];
    $deliveryDate=$_REQUEST['deliveryDate'];
    $delivered="";
    if ($deliveryCheckbox=='on')
        $delivered="YES";
    OutdoorEquipmentOrder::recordDelivery($delivered,$deliveryDate,$orderIDwanted);
    header('Location: deliveries.php');
}

?>
<p>
<form method=post action=deliveries.php>
<?
    $display=$_REQUEST['display'];
```

Save the **deliveries.php** file and copy it to the server. Open the file **OutdoorEquipmentOrder.php** and add the **recordDelivery()** method shown below. Save the file and copy it to the server.

```

public static function recordDelivery($delivered,$deliveryDate,$orderIDwanted)
{
    include('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="UPDATE outdoorEquipmentOrder SET delivered='$delivered',
        deliveryDate='$deliveryDate' WHERE orderID='$orderIDwanted'";
    $result=mysqli_query($conn, $query);
    mysqli_close($conn);
}
}
?>

```

Run the 'Plan delivery' web page. Select an order, enter a delivery date, tick the 'Delivery completed' checkbox, then click the 'update' button. The page will be reloaded with only the orders still awaiting delivery listed. Select the 'Display all orders' option. Check that the selected order is present, but now marked as delivered with the delivery date shown.

The final step is to select orders for a delivery journey. Re-open the **deliveries.php** file and add the block of program code shown below. This creates a button with the caption 'plan delivery route' and a function to determine which of the 'Select for delivery' checkboxes have been ticked. Save the **deliveries.php** file and copy it to the server.

```

echo"<input type='radio' name='display' value='awaiting'
    onclick='this.form.submit()' checked> Display orders awaiting delivery";
}
?>
</form>
<p>
<input type=button onClick=loadDeliveries(); value='plan delivery route'>
<script>
    function loadDeliveries()
    {
        var checkboxes = document.getElementsByName("select");
        var numberOfCheckedItems = 0;
        var itemString="";
        var found=false;
        for(var i = 0; i < checkboxes.length; i++)
        {
            if(checkboxes[i].checked)
            {
                if (found==true)
                {
                    itemString=itemString +",";
                }
                itemString=itemString +i;
                found=true;
            }
        }
        window.location.assign("planRoute.php?select="+itemString);
    }
</script>
<p>
<table class=stock cellpadding=10px>

```

When clicked, the 'plan delivery route' button activates a function which examines each of the checkboxes and makes up a text string with the sequence numbers of the boxes which are ticked. For example, the sequence of ticks shown below would produce the output string: 0,2,3.

OrderID	Order date	Customer	Town		Delivered	Select for delivery	Delivery
1	06-10-2019	Snowdonia Mountainsport	Betws-y-Coed	<input type="button" value="details"/>		<input checked="" type="checkbox"/>	<input type="text" value="dd/mm/yyyy"/>
2	06-10-2019	Bay Watersports	Cardiff	<input type="button" value="details"/>		<input type="checkbox"/>	<input type="text" value="dd/mm/yyyy"/>
8	06-10-2019	Pwllheli Surf Shop	Pwllheli	<input type="button" value="details"/>		<input checked="" type="checkbox"/>	<input type="text" value="dd/mm/yyyy"/>
9	06-10-2019	Lakeside Watersports	Bala	<input type="button" value="details"/>		<input checked="" type="checkbox"/>	<input type="text" value="dd/mm/yyyy"/>

Another page is then loaded, which we will now create. Open a blank file and add the program code:

```
<html>
<head>
<title> Delivery route </title>
<link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
<?
$select=$_REQUEST['select'];
echo"<p>select = ".$select;
$Aselect = explode(",", $select);
$locationTotal=count($Aselect);
include('OutdoorEquipmentOrder.php');
include('DeliveryCustomer.php');
$orderCount=OutdoorEquipmentOrder::loadOrders();
$count=0;
for ($i=1;$i<=$orderCount;$i++)
{
    $delivered = OutdoorEquipmentOrder::$orders[$i]->getDelivered();
    if ($delivered != "YES")
    {
        $found=false;
        for ($j=0;$j<$locationTotal;$j++)
        {
            if ($Aselect[$j]==$count)
                $found=true;
        }
        if ($found==true)
        {
            $orderID = OutdoorEquipmentOrder::$orders[$i]->getOrderID();
            $customerID = OutdoorEquipmentOrder::$orders[$i]->getCustomerID();
            echo"<p>orderID=".$orderID." customerID=".$customerID;
            DeliveryCustomer::loadCustomerByID($customerID);
            $companyName = DeliveryCustomer::$customerObj[1]->getCompanyName();
            $town = DeliveryCustomer::$customerObj[1]->getTown();
            $x = DeliveryCustomer::$customerObj[1]->getX();
            $y = DeliveryCustomer::$customerObj[1]->getY();
            echo", ".$companyName.", ".$town." (".$x." : ".$y.")";
        }
        $count++;
    }
}
?>
</body>
</html>
```

Save the file as **planRoute.php** and copy it to the server. This page carries out route calculations and will not be visible to the user in the final website, but we will display it for test purposes.

Run the website. Ensure that orders have been entered for about eight customers in different locations across the delivery area. Select most of the orders for delivery, leaving one or two unticked as in the example below.

Display all orders
 Display orders awaiting delivery

plan delivery route

OrderID	Order date	Customer	Town		Delivered	Select for delivery
1	06-10-2019	Snowdonia Mountainsport	Betws-y-Coed	details		<input checked="" type="checkbox"/>
2	06-10-2019	Bay Watersports	Cardiff	details		<input checked="" type="checkbox"/>
8	06-10-2019	Pwllheli Surf Shop	Pwllheli	details		<input checked="" type="checkbox"/>
9	06-10-2019	Lakeside Watersports	Bala	details		<input type="checkbox"/>
10	07-10-2019	Cader Idris Climbing Shop	Dolgellau	details		<input checked="" type="checkbox"/>
13	28-10-2019	Sailing Centre	New Quay	details		<input checked="" type="checkbox"/>
14	31-10-2019	Swansea Leisure	Swansea	details		<input checked="" type="checkbox"/>
33	14-03-2020	Lakeside Watersports	Bala	details		<input checked="" type="checkbox"/>
34	14-03-2020	Barry Island Watersports	Barry	details		<input checked="" type="checkbox"/>

Click the 'plan delivery route' button. The **planRoute.php** page should then be loaded.

The text string is displayed, made up from the sequence numbers for the ticked boxes. This information is then used to display details of the orders, including the delivery town and its map coordinates. Check that the output corresponds with the series of orders selected.

```

select = 0,1,2,4,5,6,7,8

orderID=1 customerID=7, Snowdonia Mountainsport, Betws-y-Coed (565:243)
orderID=2 customerID=6, Bay Watersports, Cardiff (791:1209)
orderID=8 customerID=16, Pwllheli Surf Shop, Pwllheli (335:355)
orderID=10 customerID=14, Cader Idris Climbing Shop, Dolgellau (533:446)
orderID=13 customerID=9, Sailing Centre, New Quay (358:767)
orderID=14 customerID=5, Swansea Leisure, Swansea (510:1094)
orderID=33 customerID=15, Lakeside Watersports, Bala (642:345)
orderID=34 customerID=32, Barry Island Watersports, Barry (752:1239)
    
```

We will now create a lookup table containing the distances in screen pixels between all possible pairs of delivery points. Return to the **planRoute.php** file and add the blocks of program code shown below.

Since the delivery van will depart from the depot in Newtown, the map coordinates of the depot are entered and this location is included in the distance table.

```

include('DeliveryCustomer.php');
$orderCount=OutdoorEquipmentOrder::loadOrders();
$count=0;

$routeCustomerID[0]=0;
$routeTown[0]='Newtown';
$routeCount=0;

for ($i=1;$i<=$orderCount;$i++)
{
    $delivered = OutdoorEquipmentOrder::$orders[$i]->getDelivered();
    if ($delivered != "YES")
    {
        . . . . .

        $x = DeliveryCustomer::$customerObj[1]->getX();
        $y = DeliveryCustomer::$customerObj[1]->getY();
        echo", ".$companyName.", ".$town." (".$x." ".$y.)";

        $routeCount++;
        $routeCustomerID[$routeCount]=$customerID;
        $routeTown[$routeCount]=$town;
    }
    $count++;
}
}

echo"<p>";
echo"<table class=stock>";
echo"<tr>";
echo"<td class=stock></td>";
for($i=0;$i<=$routeCount;$i++)
{
    echo"<td class=stock>".$routeCustomerID[$i]." ".$routeTown[$i]."</td>";
}
echo"</table>";

?>
</body>
</html>

```

Save the **planRoute.php** file and copy it to the server, then refresh the web page. Column headings for the table should be shown. Check that the set of customerIDs and delivery locations is correct.

0 Newtown	7 Betws-y-Coed	6 Cardiff	16 Pwllheli	14 Dolgellau	9 New Quay	5 Swansea	15 Bala	32 Barry
-----------	----------------	-----------	-------------	--------------	------------	-----------	---------	----------

Return to the **planRoute.php** file add lines of program code to produce row titles for the table.

```

echo"<td class=stock></td>";
for($i=0;$i<=$routeCount;$i++)
{
    echo"<td class=stock>".$routeCustomerID[$i]." ".$routeTown[$i]."</td>";
}

for($i=0;$i<=$routeCount;$i++)
{
    echo"<tr>";
    echo"<td class=stock>".$routeCustomerID[$i]." ".$routeTown[$i]."</td>";
}

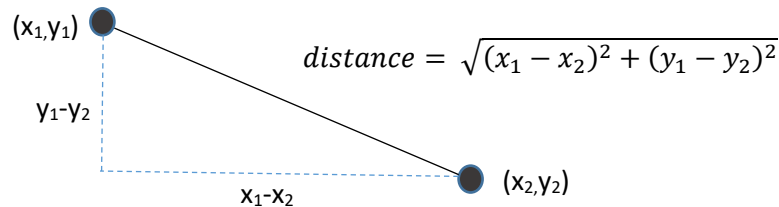
echo"</table>";
?>
</body>
</html>

```

Save **planRoute.php**, copy it to the server and refresh the web page. All table captions should now be displayed.

	0 Newtown	7 Betws-y-Coed	6 Cardiff	16 Pwllheli	14 Dolgellau	9 New Quay	5 Swansea	15 Bala	32 Barry
0 Newtown									
7 Betws-y-Coed									
6 Cardiff									
16 Pwllheli									
14 Dolgellau									
9 New Quay									
5 Swansea									
15 Bala									
32 Barry									

The next step is to calculate the journey distances between pairs of locations. This can be done easily using Pythagoras' theorem, as in the diagram below. The difference in horizontal coordinates (x_1-x_2) and the difference in vertical coordinates (y_1-y_2) are found. These values are then used to calculate the straight line distance in screen pixels between the points by means of the formula:



Return to the **planRoute.php** file. Add the lines of code shown in the boxes below.

```

$orderCount=OutdoorEquipmentOrder::loadOrders();
$count=0;
$routeCustomerID[0]=0;
$routeTown[0]='Newtown';

$routeX[0]=738;
$routeY[0]=585;

$routeCount=0;
for ($i=1;$i<=$orderCount;$i++)
{
    .....
    $x = DeliveryCustomer::$customerObj[1]->getX();
    $y = DeliveryCustomer::$customerObj[1]->getY();
    echo", ".$companyName.", ".$town." ( ".$x." : ".$y." )";
    $routeCount++;
    $routeCustomerID[$routeCount]=$customerID;
    $routeTown[$routeCount]=$town;

    $routeX[$routeCount]=$x;
    $routeY[$routeCount]=$y;
}
$count++;
}
}
echo"<p>";
echo"<table class=stock>";
echo"<tr>";
echo"<td class=stock>";
    
```


Continuing to work in the **planRoute.php** file, add the further block of code shown below.

```

for($i=0;$i<=$routeCount;$i++)
{
    echo"<tr>";
    echo"<td class=stock>".$routeCustomerID[$i]."   ".$routeTown[$i]."</td>";

    for ($j=0;$j<=$routeCount;$j++)
    {
        $Xdifference = $routeX[$i]-$routeX[$j];
        $Ydifference = $routeY[$i]-$routeY[$j];
        $distance = $Xdifference*$Xdifference+$Ydifference*$Ydifference;
        $distance = sqrt($distance);
        echo"<td class=stock>".number_format($distance,0);
        $link[$i][$j]=$distance;
    }
}
echo"</table>";

```

The blocks of code begin by providing the map coordinates for the depot, then make use of the loop to create **\$routeX[]** and **\$routeY[]** arrays which contain the map coordinates for all delivery points. The depot itself is treated as point 0.

Nested loops then operate to access each cell of the table. Straight line distances in screen pixels are calculated using Pythagoras' formula, then displayed in the table. Points on the principal diagonal will, of course, show zero distances as the start and finish locations are the same.

Save the **planRoute.php** file and copy it to the server. Refresh the page and check that a table of distances is displayed, as in this example.

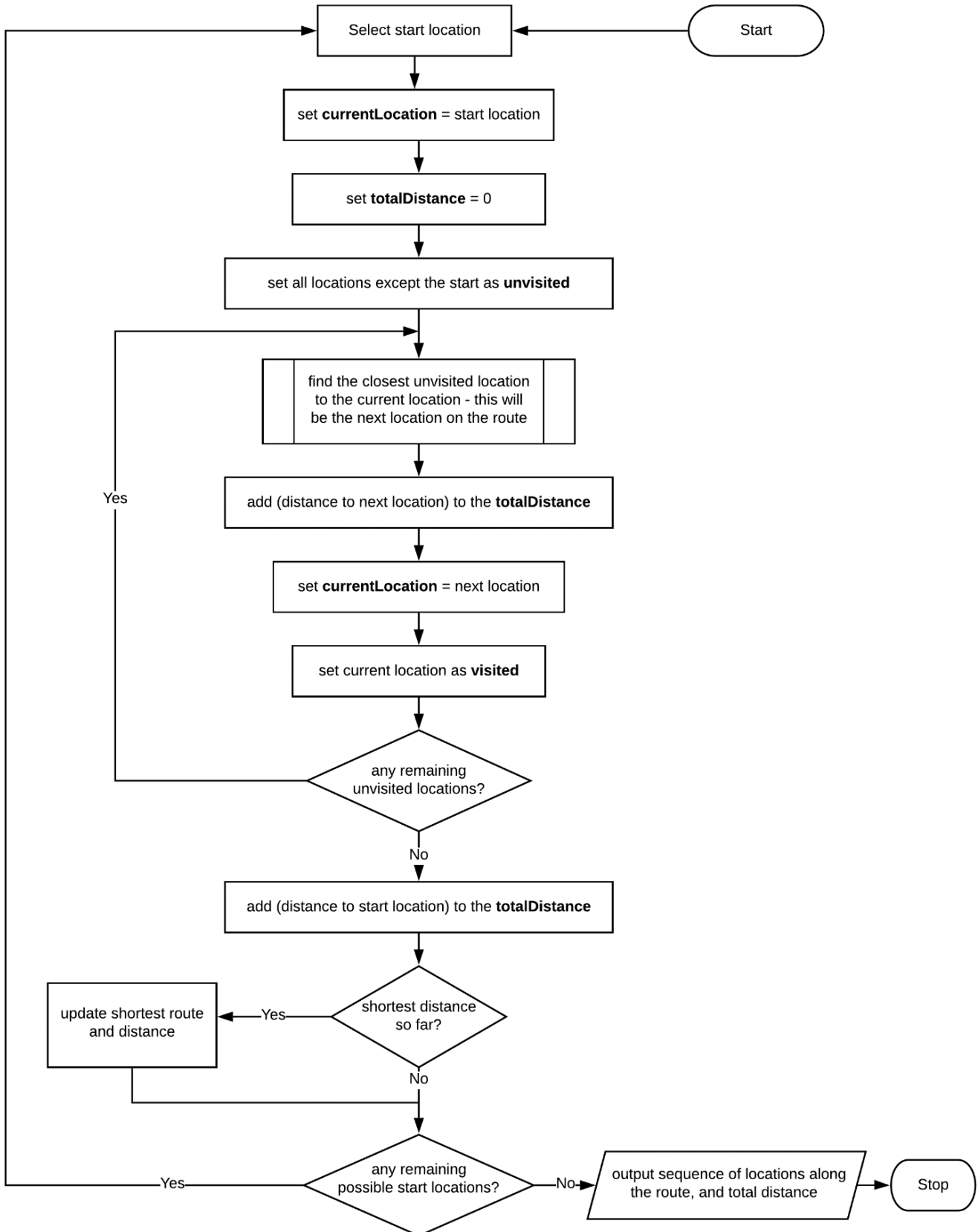
	0 Newtown	7 Betws-y-Coed	6 Cardiff	16 Pwllheli	14 Dolgellau	9 New Quay	5 Swansea	15 Bala	32 Barry
0 Newtown	0	383	626	464	248	421	558	258	654
7 Betws-y-Coed	383	0	992	256	206	563	853	128	1,013
6 Cardiff	626	992	0	968	805	619	304	877	49
16 Pwllheli	464	256	968	0	218	413	759	307	977
14 Dolgellau	248	206	805	218	0	366	648	149	823
9 New Quay	421	563	619	413	366	0	361	509	615
5 Swansea	558	853	304	759	648	361	0	761	282
15 Bala	258	128	877	307	149	509	761	0	901
32 Barry	654	1,013	49	977	823	615	282	901	0

In the next stage of the program, we will make use of the **Nearest Neighbour algorithm** to find the shortest route which leaves the depot, visits all delivery points and then returns to the depot. The algorithm sequence is:

- Choose a starting point. This is the **CURRENT** point and is recorded as 'visited'.
- Initialise the total journey distance to zero.
- REPEAT
 - Select the nearest point on the route to the **CURRENT** point which has not yet been visited. This becomes the new **CURRENT** point.
 - Add the distance for this leg of the journey.
 - Record that the **CURRENT** point has now been visited.
- UNTIL every point on the route has been visited.
- Add the direct distance from the **CURRENT** point back to the starting point to close the loop.

This algorithm will generally produce a satisfactory solution for any starting point along the route. However, this may not be the shortest route possible. To be sure of finding the shortest journey, it is necessary to repeat the algorithm for each of the possible starting points.

A program design to implement the algorithm is shown in the flow chart.



Return to the **planRoute.php** file and add the block of program code shown in the two boxes below, then save the **planRoute.php** file and copy it to the server.

```

        echo"<td class=stock>".number_format($distance,0);
        $link[$i][$j]=$distance;
    }
}
echo"</table>";

for ($start=0; $start<=$routeCount; $start++)
{
    $current = $routeCustomerID[$start];
    for ($i=0;$i<=$routeCount;$i++)
    $visited[$i]=false;
    $visited[$start]=true;
    $finished=false;
    $totalDistance=0;
    $pointCount=0;
    while ($finished==false)
    {
        echo"<p>current = ".$current;
        for ($j=0;$j<=$routeCount;$j++)
        {
            if ($routeCustomerID[$j]==$current)
            {
                echo" - ".$routeTown[$j]." - REMAINING: ";
                $first=true;
                for ($i=0;$i<=$routeCount;$i++)
                {
                    if ($visited[$i]==false)
                    {
                        echo"   ".$routeTown[$i].
                            "(" .number_format($link[$i][$j],0).")";
                        if ($first==true)
                        {
                            $min=$link[$i][$j];
                            $next=$i;
                            $first=false;
                        }
                        else
                        {
                            if ($link[$i][$j]<$min)
                            {
                                $min=$link[$i][$j];
                                $next=$i;
                            }
                        }
                    }
                }
                $totalDistance=$totalDistance+$min;
            }
        }
        $current=$routeCustomerID[$next];
        $visited[$next]=true;
        $finished=true;
    }
}

```

```

        $totalDistance=$totalDistance+$min;
    }
}
$current=$routeCustomerID[$next];
$visited[$next]=true;
$finished=true;

for ($i=0;$i<=$routeCount;$i++)
{
    if ($visited[$i]==false)
        $finished=false;
}
}
echo"<p>current = ".$current." - ".$routeTown[$next];
$returnDistance = $link[$next][$start];
echo"<p>Return distance to ".$routeTown[$start].
        " = ".number_format($returnDistance,0);
$totalDistance=$totalDistance+$returnDistance;
echo"<p>TOTAL DISTANCE = ".number_format($totalDistance,0);
echo"<p><hr><p>";
}
?>
</body>
</html>

```

Refresh the **planRoute.php** page and examine the output. The program implements the Nearest Neighbour algorithm for each of the possible starting points around the delivery route.

The first block of output shown below begins at the Newtown depot (location 0). At each stage, a list is given of the remaining unvisited locations and their distances in screen pixels from the current location. The program then selects the closest point to become the new current location. This procedure continues until all points have been visited, then the distance back to the starting point is added. Carefully check that the algorithm has been performed correctly for your test data, for each of the possible starting points.

```

current = 0 - Newtown - REMAINING: Betws-y-Coed(383) Cardiff(626) Pwllheli(464) Dolgellau(248) New Quay(421) Swansea(558) Bala(258) Barry(654)
current = 14 - Dolgellau - REMAINING: Betws-y-Coed(206) Cardiff(805) Pwllheli(218) New Quay(366) Swansea(648) Bala(149) Barry(823)
current = 15 - Bala - REMAINING: Betws-y-Coed(128) Cardiff(877) Pwllheli(307) New Quay(509) Swansea(761) Barry(901)
current = 7 - Betws-y-Coed - REMAINING: Cardiff(992) Pwllheli(256) New Quay(563) Swansea(853) Barry(1,013)
current = 16 - Pwllheli - REMAINING: Cardiff(968) New Quay(413) Swansea(759) Barry(977)
current = 9 - New Quay - REMAINING: Cardiff(619) Swansea(361) Barry(615)
current = 5 - Swansea - REMAINING: Cardiff(304) Barry(282)
current = 32 - Barry - REMAINING: Cardiff(49)
current = 6 - Cardiff
Return distance to Newtown = 626
TOTAL DISTANCE = 2,511

```

Return to the **planRoute.php** file. It is necessary to keep a record of the most suitable route for the delivery journey.

Begin by adding a **session_start()** command at the beginning of the page. This will allow the route details to be stored for further use.

```

<?
  session_start();
?>

<html>
<head>
  <title> Delivery route </title>

```

Add an array variable `$routeCustomerWanted[0]` to record the start point for the current route.

```

$totalDistance=0;
$pointCount=0;

$routeCustomerWanted[0]=$current;

while ($finished==false)
{

```

Add the lines of program code shown below.

```

    $totalDistance=$totalDistance+$min;
  }
}
$current=$routeCustomerID[$next];
$visited[$next]=true;

$pointCount++;
$routeCustomerWanted[$pointCount]=$current;

$finished=true;
for ($i=0;$i<=$routeCount;$i++)
{
  if ($visited[$i]==false)
    . . . . .
$totalDistance=$totalDistance+$returnDistance;
echo"<p>TOTAL DISTANCE = ".number_format($totalDistance,0);
echo"<p><hr><p>";

```

```

if ($start==0)
{
  $minTotal=$totalDistance;
  $minRoute=0;
}
else
{
  if ($totalDistance<$minTotal)
  {
    $minTotal=$totalDistance;
    $minRoute=$start;
  }
}
$pointCount++;
$routeCustomerWanted[$pointCount]=$routeCustomerWanted[0];
if($minRoute==$start)
  $_SESSION["routeCustomerWanted"]=$routeCustomerWanted;
}
?>

```

```

<form method=post action='route.php'>
<input type=submit value='continue'>
</form>

</body>

```

Save the **planRoute.php** file and copy it to the server.

The additional program lines inserted above will store values in the **\$routeCustomerWanted[]** array as each journey point is found. This is an array which records the sequence of delivery points by means of the customerID values. When the route is completed, the total journey distance is checked to determine whether this is the shortest route found so far. If so, the array is stored as the **routeCustomerWanted** session variable. After all processing is completed, a 'continue' button will lead to the next web page. We will create this page now.

Open a blank file and insert the program code shown below.

```
<?
    session_start();
?>
<html>
<head>
    <title> Delivery route </title>
    <link rel="stylesheet" type="text/css" href="styleSheet.css" />
    <script src="p5.js"></script>
    <script src="p5.dom.js"></script>
</head>
<body>
    <?
        include('staffMenu.php');
        include('DeliveryCustomer.php');
        $routeCustomerWanted = $_SESSION["routeCustomerWanted"];
        $locationTotal=count($routeCustomerWanted);
        for ($i=0; $i<$locationTotal; $i++)
        {
            $customerIDwanted = $routeCustomerWanted[$i];
            echo"<br>point ".$i.".": ".$customerIDwanted;
            if ($customerIDwanted==0)
            {
                $town[$i]='Newtown';
                $x[$i]=738;
                $y[$i]=585;
            }
            else
            {
                DeliveryCustomer::loadCustomerByID($customerIDwanted);
                $town[$i]=DeliveryCustomer::$customerObj[1]->getTown();
                $x[$i]=DeliveryCustomer::$customerObj[1]->getX();
                $y[$i]=DeliveryCustomer::$customerObj[1]->getY();
            }
            echo": ".$town[$i]." (".$x[$i].", ".$y[$i].")";
        }
    ?>
</body>
</html>
```

Save the file as **route.php** and copy it to the server. Run the website, select orders for delivery then move on to calculate the shortest delivery route. Click the 'continue' button after the calculations are completed. A new page opens, listing the order in which customers will be visited. Notice that the sequence ends by repeating the first point specified, so that a closed loop is formed. In practice, the journey would begin and end at the Newtown depot.

```
point 0: 9: New Quay (358, 767)
point 1: 5: Swansea (510, 1094)
point 2: 32: Barry (752, 1239)
point 3: 6: Cardiff (791, 1209)
point 4: 0: Newtown (738, 585)
point 5: 14: Dolgellau (533, 446)
point 6: 15: Bala (642, 345)
point 7: 7: Betws-y-Coed (565, 243)
point 8: 16: Pwllheli (335, 355)
point 9: 9: New Quay (358, 767)
```

Return to the **route.php** file and add the blocks of p5.js program code shown below. These create a scrolling map, similar to the map used earlier for the input of customers.

```

        $y[$i]=DeliveryCustomer::$customerObj[1]->getY();
    }
    echo": ".$town[$i]." (".$x[$i].", ".$y[$i].")";
}
?>

<p>
<script>
var scrollPosition=0;
var vPos;
var ratio;
var scrollSelected = false;

function preload()
{
    img1=loadImage("map.png");
}

function setup()
{
    createCanvas(1000, 640);
}

function draw()
{
    ratio = 1260/640;
    tv = map(scrollPosition, 0, height, 0, 1264-height);
    translate(0, -tv);
    image(img1, 0, 0);
    scrollbar(scrollPosition);
    x=mouseX;
    y=mouseY;
    if (x>=960)
    {
        fill(0);
        rect(986,vPos,14,14);
        if (mouseIsPressed==true)
        {
            scrollPosition=y;
            if (scrollPosition<0)
            {
                scrollPosition=0;
            }
            if (scrollPosition>640)
            {
                scrollPosition=640;
            }
        }
    }
}
}
</script>
</body>
</html>

```

Add the **scrollbar()** method shown below.

```

function scrollbar(scrollPosition)
{
    noStroke();
    fill(204);
    rect(986,0,14,1264);
    if (scrollSelected==false)
    {
        fill(102);
    }
    else
    {
        fill(40);
    }
    vPos = scrollPosition * ratio;
    rect(986,vPos,14,14);
}

</script>
</body>
</html>

```

Save the **route.php** file and copy it to the server. Refresh the page. Check that the map is displayed in a window, and can be scrolled by means of the bar at the right of the window.

Return to the **route.php** file. The next step is to display the delivery route on the map. Begin by adding lines of program code to convert PHP arrays to JavaScript arrays, so that customer IDs and map coordinates for points along the route are available in p5.js.

```

    $y[$i]=DeliveryCustomer::$customerObj[1]->getY();
}
echo": ".$town[$i]." (".$x[$i].", ".$y[$i].")";
}
?>
<p>
<script>

```

```

var routeX = <? echo json_encode($x) ?>;
var routeY = <? echo json_encode($y) ?>;
var routeCustomer = <? echo json_encode($routeCustomerWanted) ?>;
var locationTotal=<? echo $locationTotal ?>;

```

```

var scrollPosition=0;
var vPos;
var ratio;
var scrollSelected = false;

function preload()
{
    img1=loadImage("map.png");
}

```

We will now add a **drawRoute()** method to the **<script>** block after the **draw()** method. This new method contains a loop which will operate for each of the points listed in the delivery route. Red circular markers are placed at each of the delivery points, and the points along the route are connected by straight lines.

Add a line of code at the end of the **draw()** method, as shown below, to call **drawRoute()** when the screen display is refreshed.

```

        if (scrollPosition>640)
        {
            scrollPosition=640;
        }
    }
}

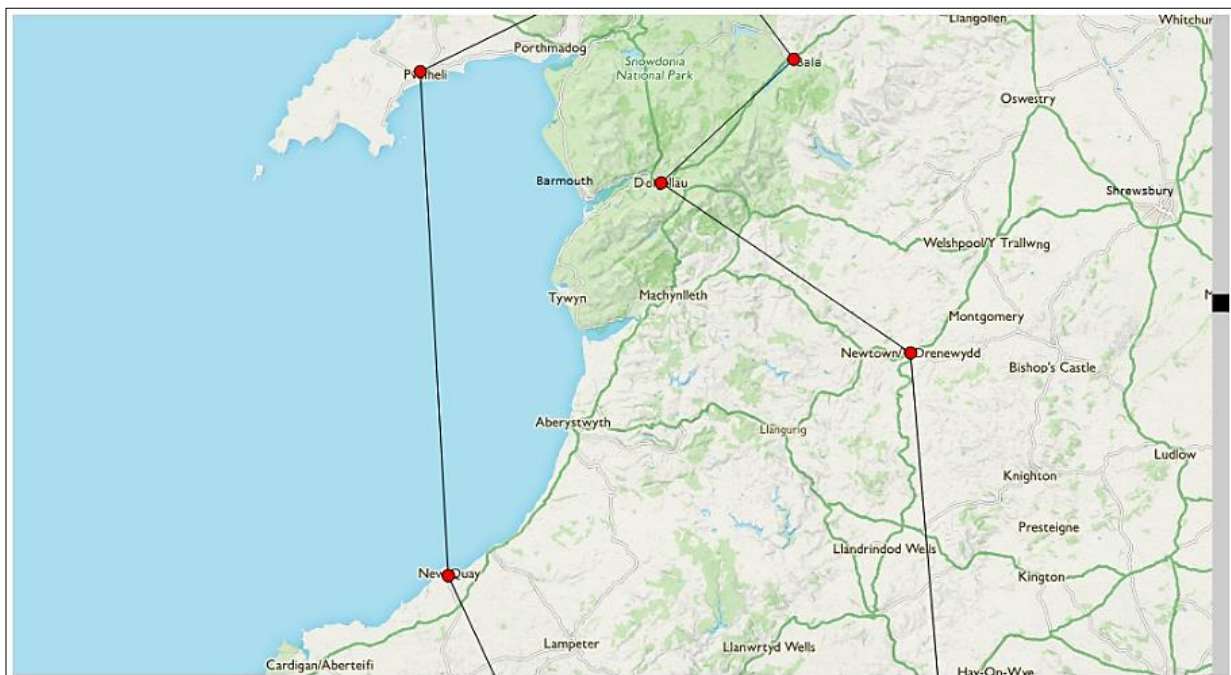
drawRoute();
}

function drawRoute()
{
    for (var i=0; i<(locationTotal-1); i++)
    {
        stroke(0);
        line(routeX[i],routeY[i],routeX[i+1],routeY[i+1]);
        fill(255,0,0);
        ellipse(routeX[i],routeY[i],10,10);
    }
    fill(255,0,0);
    ellipse(routeX[0],routeY[0],10,10);
}

function scrollbar(scrollPosition)
{

```

Save the **route.php** file, copy it to the server, then refresh the page. The route should now be displayed on the map as shown on the page below, and can be viewed in full by scrolling the map up and down. Check that the delivery points are in the correct positions on the map and have been linked in the correct sequence.



Whilst a correct sequence of points has been plotted, the route is clearly oversimplified - with one of the links crossing the sea! In other places, the route does not follow the actual road pattern with sufficient accuracy. We can resolve this problem by allowing additional points to be added between customer locations:

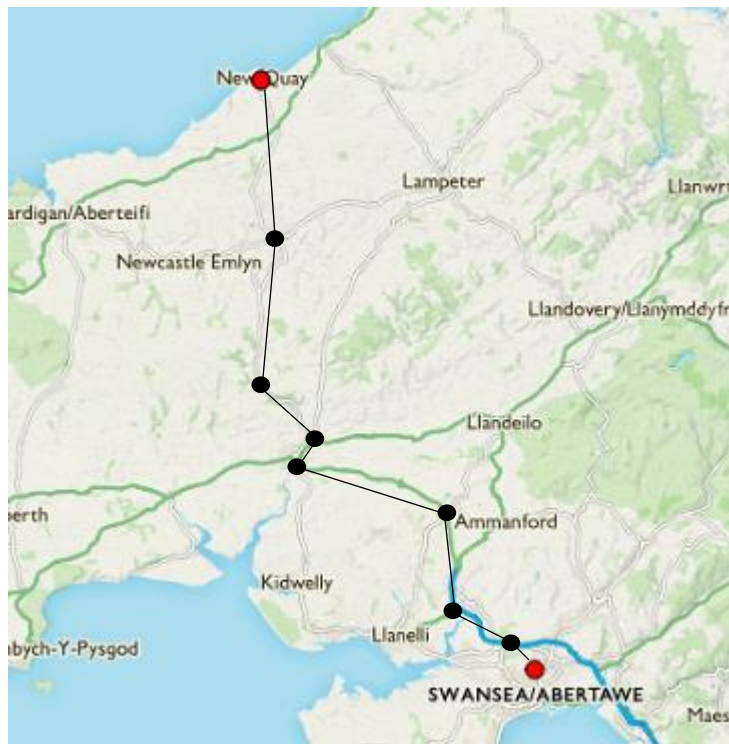
The user will be able to add a route point to any of the links.



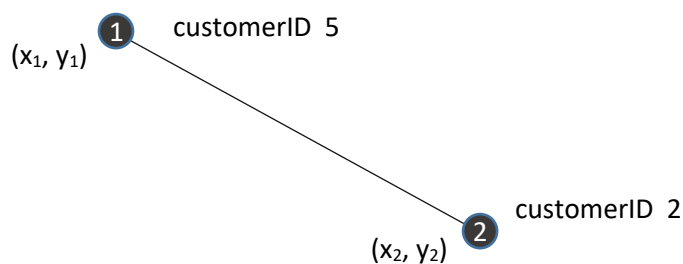
The new route point may then be dragged to the required position on the road network.



Additional route points may be added and dragged into position as necessary, until the link between delivery locations has been defined with sufficient accuracy.



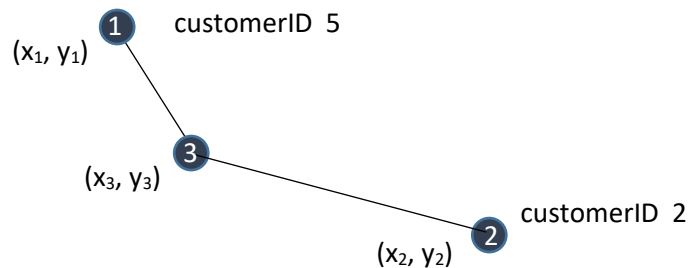
The routes between delivery points can be stored in a database table in the form of a **linked list**. Let us suppose that the first link of the delivery journey is a straight line segment linking the customer identified by **customerID 5**, and the customer identified by **customerID 2**. The x and y map coordinates of each of the route points are known.



A record for each point is inserted in a **roadPoints** table. The **customerID** values are recorded for only the starting point of the link, with zero values inserted for the end point. The map coordinates are then added for both points. Finally, a pointer in the start record links to the destination record, in this case with **roadPointID 2**. The pointer at the destination is set to -1.

roadPointID	fromCustomer	toCustomer	xpos	ypos	pointer
start 1	5	2	x_1	y_1	2
2	0	0	x_2	y_2	-1

The linked list structure can easily allow for the addition of intermediate points in the link.



If another point is added, a record is inserted in the next available location in the table: in this case at **roadPointID 3**. The pointers are adjusted to link the points in the correct sequence from the start to the destination.

roadPointID	fromCustomer	toCustomer	xpos	ypos	pointer
start 1	5	2	x_1	y_1	3
2	0	0	x_2	y_2	-1
3	0	0	x_3	y_3	2

The pointer from record 1 now links to record 3, and the pointer from record 3 links to record 2 which is the destination. The end of the sequence is marked by a -1 pointer value.

It will be convenient to also include a **backpointer**, which will allow the sequence of points to be followed in reverse order. The backpointer values will lead from any point to the start of the link, where the sequence is terminated by a -1 value. For example, beginning at **roadPointID 2**:

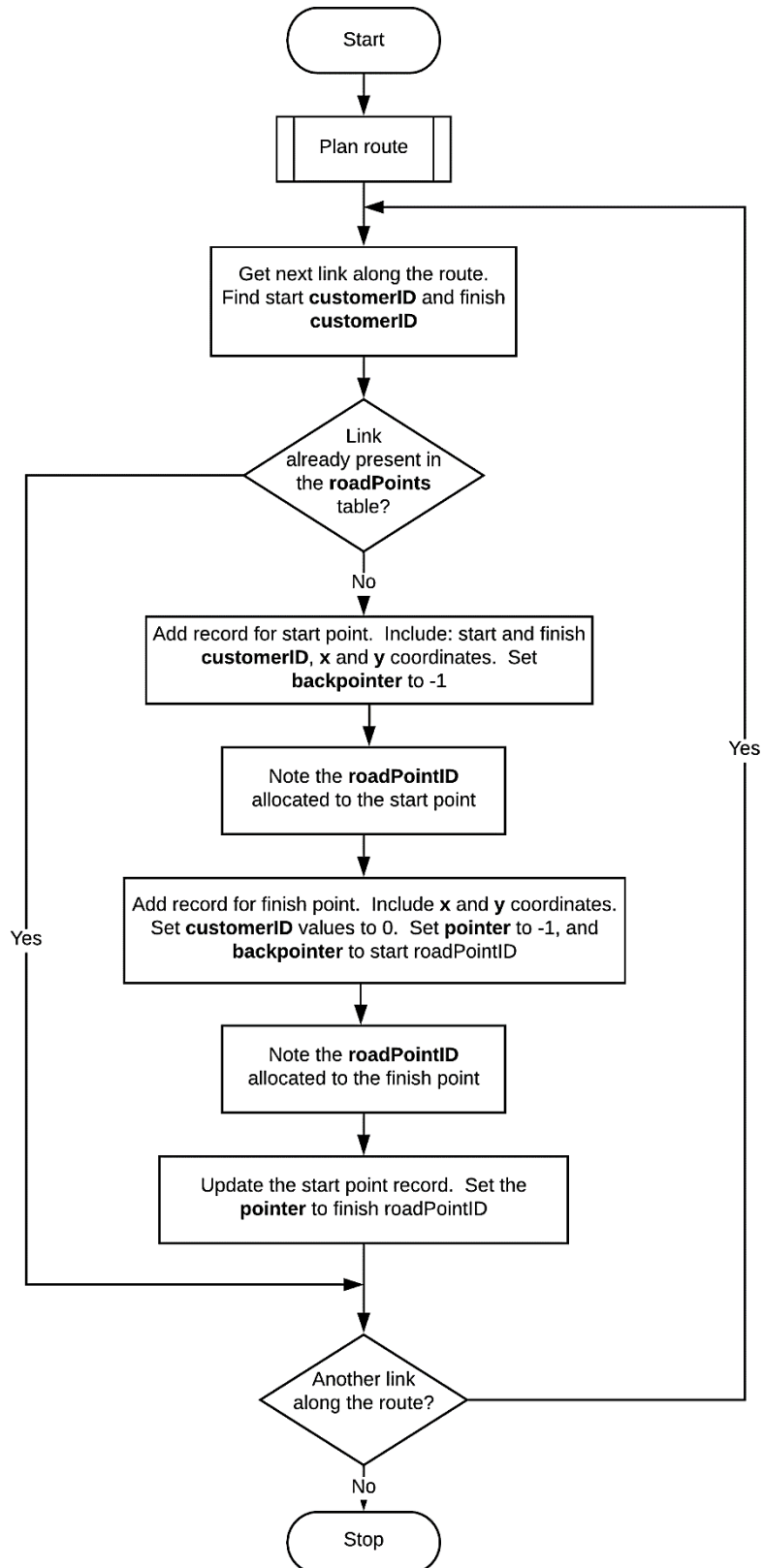
roadPointID	fromCustomer	toCustomer	xpos	ypos	pointer	backpointer
1	5	2	x_1	y_1	3	-1
start 2	0	0	x_2	y_2	-1	3
3	0	0	x_3	y_3	2	1

The pointer at record 2 now links to record 3, and the pointer at record 3 now links to record 1 which was the start point.


We can set up a procedure for initialising the linked lists in a **roadPoints** table. Each link of the delivery route will require two **roadPoint** records representing the start and finish points, as in the example below. The records are connected forwards and backwards by the pointers.

roadPointID	fromCustomer	toCustomer	xpos	ypos	pointer	backpointer
start 1	5	2	x_1	y_1	2	-1
2	0	0	x_2	y_2	-1	1

The flowchart illustrates the steps required. The route will first be calculated by means of the Nearest Neighbour algorithm. Each link between a pair of delivery points along the route is then considered. A linked list connecting the points may already be present in the road point table. If not, a record pair will be created. Pointer values are set using the **roadPointID** values which are allocated by the computer as auto-numbers when the records are added to the table.



To implement the linked list, we will begin by creating the database table. Go to the PHP MyAdmin page, list the tables and select the 'new' option. Set up a table with the name **roadPoints** and add the fields shown below. All fields have the data type **integer**. The **roadPointID** field is identified as the primary key, and is set to auto-increment as records are added.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	roadPointID 	int(11)			No	None		AUTO_INCREMENT
2	fromCustomer	int(11)			No	None		
3	toCustomer	int(11)			No	None		
4	xpos	int(11)			No	None		
5	ypos	int(11)			No	None		
6	pointer	int(11)			No	None		
7	backpointer	int(11)			No	None		

We will create a **RoadPoints** class to handle the set of linked lists. Open a blank file and add the lines of program code below.

The class begins by defining the attributes for a **RoadPoint** object. These correspond with the fields of the database table. Notice that the attributes have been defined as **public**, rather than private. This is to allow the objects to be translated into JavaScript by means of JSON encoding, as we did previously in the Caravan Park project in chapter 4. A **constructor** method is then added.

```
<?
class RoadPoints
{
    public static $location = array();
    public $roadPointID;
    public $fromCustomer;
    public $toCustomer;
    public $xpos;
    public $ypos;
    public $pointer;
    public $backpointer;

    function __construct($roadPointID,$fromCustomer,$toCustomer,
                        $xpos,$ypos,$pointer,$backpointer)
    {
        $this->roadPointID = $roadPointID;
        $this->fromCustomer = $fromCustomer;
        $this->toCustomer = $toCustomer;
        $this->xpos = $xpos;
        $this->ypos = $ypos;
        $this->pointer = $pointer;
        $this->backpointer = $backpointer;
    }
}
?>
```

Add the **checkStartFinish()** method shown below.

The **checkStartFinish()** method will scan through the database table to determine whether a linked list joining the specified pair of delivery points already exists in the table. If so, the **roadPointID** at the start of the list will be returned. If not, a result of 0 is returned.

```

public static function checkStartFinish($startID, $finishID)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM roadPoints WHERE fromCustomer='$startID'
                                                AND toCustomer='$finishID'";

    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    if ($num==0)
    {
        return 0;
    }
    else
    {
        $row=mysqli_fetch_assoc($result);
        $roadPointID=$row["roadPointID"];
        return $roadPointID;
    }
}
}
?>

```

We will now move on to create a new linked list pair if none is found during the search. Add the **createLink()** method shown below to the RoadPoint class file. This stores two records, one for the starting point and one for the destination of the link. It is necessary to return to the start record to update the **pointer** value once the destination record has been created, and its RoadPointID has been allocated by the computer.

```

public static function createLink($startID,$finishID,$startX,
                                $startY,$finishX,$finishY)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="INSERT INTO roadPoints VALUES ('','$startID','$finishID',
                                          '$startX','$startY','0','-1')";

    $result=mysqli_query($conn, $query);
    $startRoadPointID = mysqli_insert_id($conn);
    $query="INSERT INTO roadPoints VALUES ('','0','0','$finishX',
                                          '$finishY','-1','$startRoadPointID')";

    $result=mysqli_query($conn, $query);
    $finishRoadPointID = mysqli_insert_id($conn);
    $query="UPDATE roadPoints SET pointer='$finishRoadPointID'
          WHERE roadPointID = '$startRoadPointID'";

    $result=mysqli_query($conn, $query);
    mysqli_close($conn);
}
}
?>

```

Save the class file as **RoadPoints.php** and copy it to the server.

Return to the **route.php** page file. Add lines of program code as shown below. Each step of the delivery route is considered in turn. The **checkStartFinish()** method in the **RoadPoints** class is called to check whether a linked list already exists for this step of the route. If not, the **createLink()** method is called to produce a linked list pair.

```

    DeliveryCustomer::loadCustomerByID($customerIDwanted);
    $town[$i]=DeliveryCustomer::$customerObj[1]->getTown();
    $x[$i]=DeliveryCustomer::$customerObj[1]->getX();
    $y[$i]=DeliveryCustomer::$customerObj[1]->getY();
}
echo": ".$town[$i]." ( ".$x[$i].", ".$y[$i].")";
}
}
include('RoadPoints.php');
for ($i=0; $i<($locationTotal-1); $i++)
{
    $start = $routeCustomerWanted[$i];
    $finish = $routeCustomerWanted[$i+1];
    $result = RoadPoints::checkStartFinish($start,$finish);
    if ($result==0)
    {
        RoadPoints::createLink($start,$finish,$x[$i],$y[$i],$x[$i+1],$y[$i+1]);
    }
}
?>
<p>
<script>

```

Save the **route.php** file and copy it to the server. Run the website and go to the 'Plan delivery' page. Select a set of orders for delivery, then continue to the route calculation and map display. The sequence of delivery points will be listed.

Go now to the PHP MyAdmin website and open the roadPoints table. Check that a series of linked list pairs have been created which correspond with the steps of the selected delivery route, as in the example below.

roadPointID	fromCustomer	toCustomer	xpos	ypos	pointer	backpointer
1	9	5	358	767	2	-1
2	0	0	510	1094	-1	1
3	5	32	510	1094	4	-1
4	0	0	752	1239	-1	3
5	32	6	752	1239	6	-1
6	0	0	791	1209	-1	5

The next task is to adapt the program so that the route is drawn from the data contained in the roadPoints table.

Re-open the **RoadPoints.php** file and add the **loadRoadPoints()** method shown below. This accesses the roadPoints table in the database and creates a set of RoadPoint objects. Save **RoadPoints.php** and copy it to the server.


```

public static function loadRoadPoints()
{
    include('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM roadPoints";
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    for ($i=1;$i<=$num; $i++)
    {
        $row=mysqli_fetch_assoc($result);
        $roadPointID=$row["roadPointID"];
        $fromCustomer=$row["fromCustomer"];
        $toCustomer=$row["toCustomer"];
        $xpos=$row["xpos"];
        $ypos=$row["ypos"];
        $pointer=$row["pointer"];
        $backpointer=$row["backpointer"];
        $obj = new RoadPoints($roadPointID,$fromCustomer,
                             $toCustomer,$xpos,$ypos,$pointer,$backpointer);
        RoadPoints::$location[$i] = $obj;
    }
    return $num;
}
}
?>

```

Return to the **route.php** file and locate the **drawRoute()** function. Replace the **line()** command with a call to a new function **plotLink()** as shown below.

```

function drawRoute()
{
    for (var i=0; i<(locationTotal-1); i++)
    {
        stroke(0);
        plotLink(routeCustomer[i], routeCustomer[i+1]);
        fill(255,0,0);
        ellipse(routeX[i],routeY[i],10,10);
    }
    fill(255,0,0);
    ellipse(routeX[0],routeY[0],10,10);
}

```

Add the **plotLink()** function immediately underneath **drawRoute()** method, as shown below. The function begins by checking each of the RoadPoint objects, to see if the start and finish customerID values are present as attributes. If so, this RoadPoint object will be the start of a linked list of points joining the required customers. The function then uses the pointer values to follow the sequence of road points, drawing line segments for each, until the end of the linked list is reached.


```

fill(255,0,0);
ellipse(routeX[0],routeY[0],10,10);
}

function plotLink(start,finish)
{
  for (i=1;i<pointCount; i++)
  {
    point1=roadPoints[i].fromCustomer;
    point2=roadPoints[i].toCustomer;
    if (((point1==start)&&(point2==finish))
        ||((point2==start)&&(point1==finish)))
    {
      current = i;
    }
  }
  finished=false;
  while (finished==false)
  {
    x1=roadPoints[current].xpos;
    y1=roadPoints[current].ypos;
    next=roadPoints[current].pointer;
    x2=roadPoints[next].xpos;
    y2=roadPoints[next].ypos;
    line(x1,y1,x2,y2);
    if (roadPoints[next].pointer==-1)
    {
      finished=true;
    }
    current=next;
  }
}

function scrollbar(scrollPosition)
{
  noStroke();

```

Go now to the start of the **<script>** block and add PHP and JavaScript variables, as shown below.

```

for ($i=0; $i<($locationTotal-1); $i++)
{
  $start = $routeCustomerWanted[$i];
  $finish = $routeCustomerWanted[$i+1];
  $result = RoadPoints::checkStartFinish($start,$finish);
  if ($result==0)
  {
    RoadPoints::createLink($start,$finish,$x[$i],$y[$i],$x[$i+1],$y[$i+1]);
  }
}

$pointCount = RoadPoints::loadRoadPoints();
$roadPointsJSON = json_encode(RoadPoints::$location);
?>
<p>
<script>

var roadPoints = <? echo $roadPointsJSON ?>;
var pointCount = <? echo $pointCount ?>;

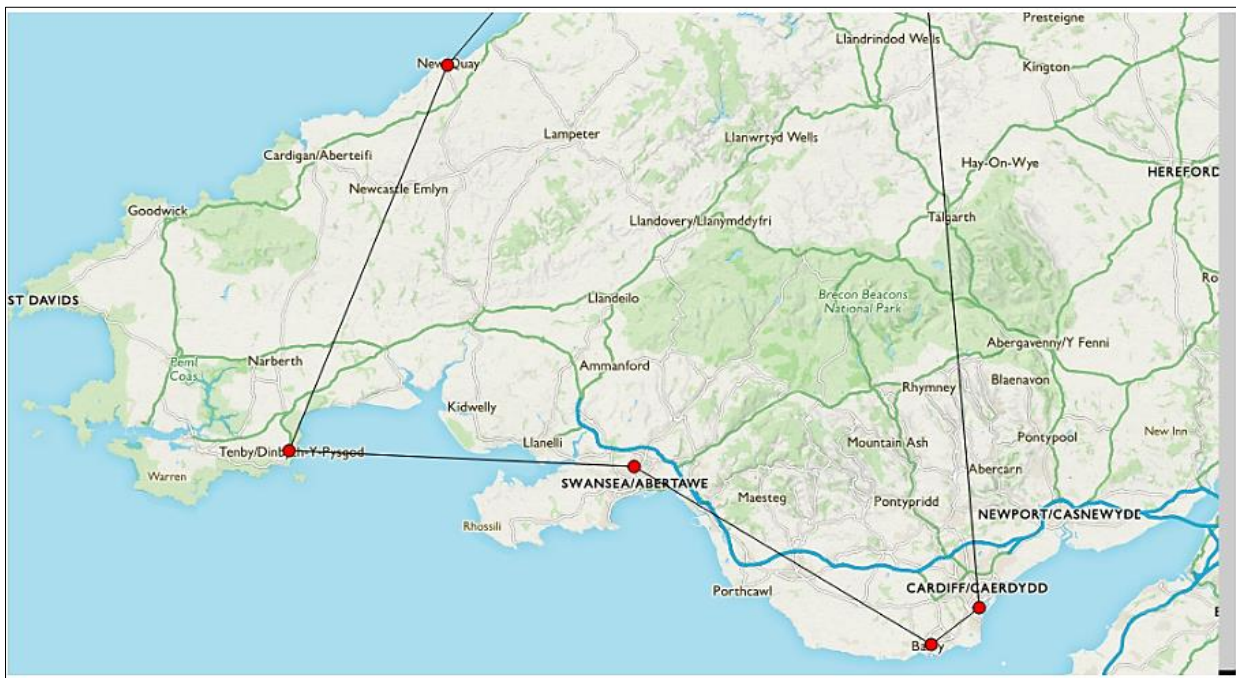
var routeX = <? echo json_encode($x) ?>;
var routeY = <? echo json_encode($y) ?>;

```

The RoadPoint data is loaded as a set of PHP objects in JavaScript Object Notation (JSON), then converted to an equivalent set of JavaScript objects for use in drawing the route map.

	PHP	JavaScript
array of objects	\$location[]	roadPoints[]
attributes	\$location[]-> roadPointID	roadPoints[].roadPointID
	\$location[]->fromCustomer	roadPoints[].fromCustomer
	\$location[]->toCustomer	roadPoints[].toCustomer
	\$location[]->xpos	roadPoints[].xpos
	\$location[]->ypos	roadPoints[].ypos
	\$location[]->pointer	roadPoints[].pointer
	\$location[]->backpointer	roadPoints[].backpointer

Save the **route.php** file and copy it to the server. Run the website and select a slightly different set of delivery customers. Continue to the route calculation and map display pages. The map should again display the delivery route, but this time it will be drawn using data loaded from the roadPoints table in the database.



Go to the PHP MyAdmin website and examine the **roadPoints** table. Existing linked lists will have been used where a pair of customers were connected during the earlier map test, but additional linked list pairs should have been added for new connections not on the previous route.

Return to the **route.php** file. The list of delivery locations near the start of the page was included only for test purposes and can now be removed. Delete the 'echo' command:

```

for ($i=0; $i<$locationTotal; $i++)
{
    $customerIDwanted = $routeCustomerWanted[$i];
    echo"<br>point ".$i." : ".$customerIDwanted;
    if ($customerIDwanted==0)
    {
        $town[$i]='Newtown';
    }
}
    
```

Delete the second 'echo' command below:

```

else
{
    DeliveryCustomer::loadCustomerByID($customerIDwanted);
    $town[$i]=DeliveryCustomer::$customerObj[1]->getTown();
    $x[$i]=DeliveryCustomer::$customerObj[1]->getX();
    $y[$i]=DeliveryCustomer::$customerObj[1]->getY();
}
echo": ".$town[$i]." (".$x[$i].", ".$y[$i].")";
}

```

REMOVE

We can now proceed to add intermediate points to the road connections. Go to the **plotLink()** function and add the lines of program code below. These begin by sorting the coordinates for the line segment, so that x1 is the smaller of the horizontal coordinates, and y1 is the smaller of the vertical coordinates. The position of the mid point on the line segment is calculated, and a black circle is displayed when the mouse pointer is close to this point.

```

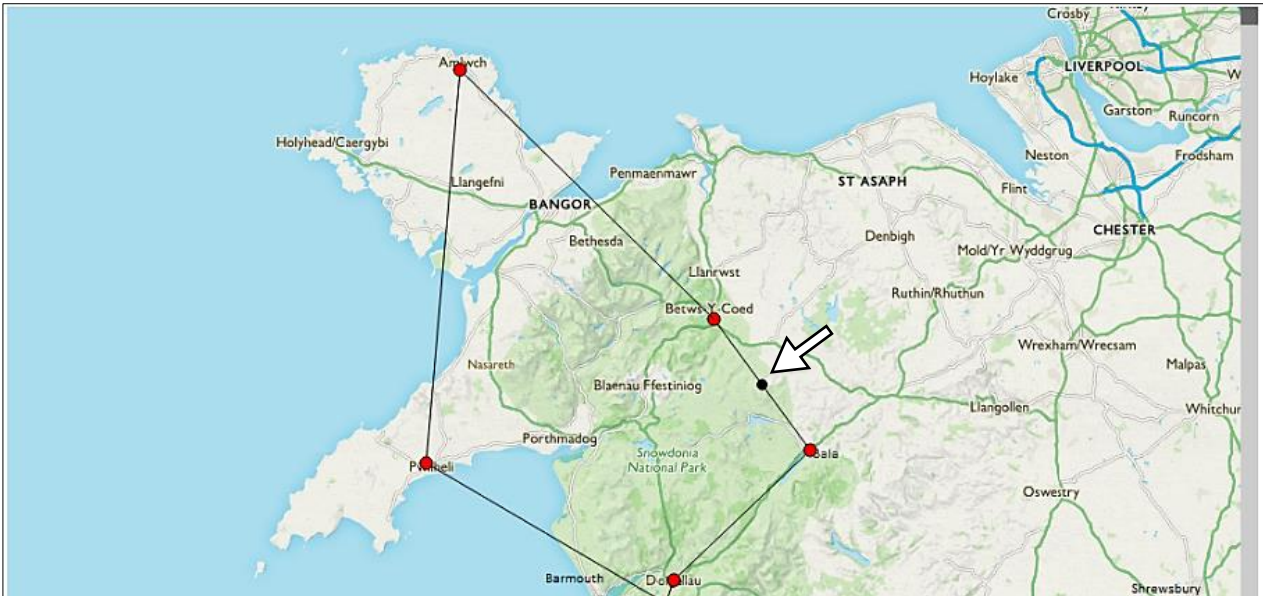
finished=false;
while (finished==false)
{
    x1=roadPoints[current].xpos;
    y1=roadPoints[current].ypos;
    next=roadPoints[current].pointer;
    x2=roadPoints[next].xpos;
    y2=roadPoints[next].ypos;
    line(x1,y1,x2,y2);

    x=mouseX;
    y=mouseY;
    y=y+tv;
    if (int(x1)>int(x2))
    {
        temp=x1;
        x1=x2;
        x2=temp;
    }
    if (int(y1)>int(y2))
    {
        temp=y1;
        y1=y2;
        y2=temp;
    }
    midX=(int(x1)+int(x2))/2;
    midY=(int(y1)+int(y2))/2;
    if ((Math.abs(x-midX)<10)&&(Math.abs(y-midY)<10))
    {
        fill(0);
        ellipse(midX,midY,8,8);
        fill(255,0,0);
    }

    if (roadPoints[next].pointer==-1)
    {
        finished=true;
    }
    current=next;
}
}

```

Save **route.php** and copy it to the server. Refresh the web page.
 When the mouse is moved close to the mid point of any link, a black circle should appear.



We can now work on the program to add the new point to the **roadPoints** linked list connecting the pair of customers.

Return to the **plotLink()** function and add the lines of program code below. This will call a **loadPage()** function if the mouse button is pressed while the pointer is on the black circle.

```

midX=(int(x1)+int(x2))/2;
midY=(int(y1)+int(y2))/2;
if ((Math.abs(x-midX)<10)&&(Math.abs(y-midY)<10))
{
    fill(0);
    ellipse(midX,midY,8,8);
    fill(255,0,0);

    if (mouseIsPressed==true)
    {
        loadPage();
    }
}
if (roadPoints[next].pointer== -1)
{
    finished=true;
}
current=next;
}
    
```

Insert the **loadPage()** function immediately after the **plotLink()** function, as show below.

The **loadPage()** function displays a confirm dialogue box. The user may click 'OK' to add the additional point to the route, or click 'cancel'. In either case, the **route.php** page will be reloaded.

Insert the `loadPage()` function immediately below the `plotLink()` function.

```

    if (roadPoints[next].pointer===-1)
    {
        finished=true;
    }
    current=next;
}
}

function loadPage()
{
    choice = confirm('add another point');
    if (choice===true)
    {
        window.location = "addPoint.php?scrollPosition="+scrollPosition
            +"&current="+current+"&next="+next+"&midX="+midX+"&midY="+midY;
    }
    else
        window.location ="route.php?scrollPosition="+scrollPosition;
}

function scrollbar(scrollPosition)
{
    noStroke();
    fill(204);
    rect(986,0,14,1264);
}

```

A few changes to the program will be necessary to ensure that the reloading works correctly. Begin by adding lines of code to the `<head>` section of `route.php`.

```

<head>
  <title> Delivery route </title>
  <link rel="stylesheet" type="text/css" href="styleSheet.css" />
  <script src="p5.js"></script>
  <script src="p5.dom.js"></script>

  <?
    $scrollPosition=$_REQUEST['scrollPosition'];
    if (isset($scrollPosition)==false)
    {
        $scrollPosition= 0;
    }
  ?>

</head>

```

Go now to the start of the `<script>` block. Replace the `scrollPosition` line:

```

var routeX = <? echo json_encode($x) ?>;
var routeY = <? echo json_encode($y) ?>;
var routeCustomer = <? echo json_encode($routeCustomerWanted) ?>;
var locationTotal=<? echo $locationTotal ?>;

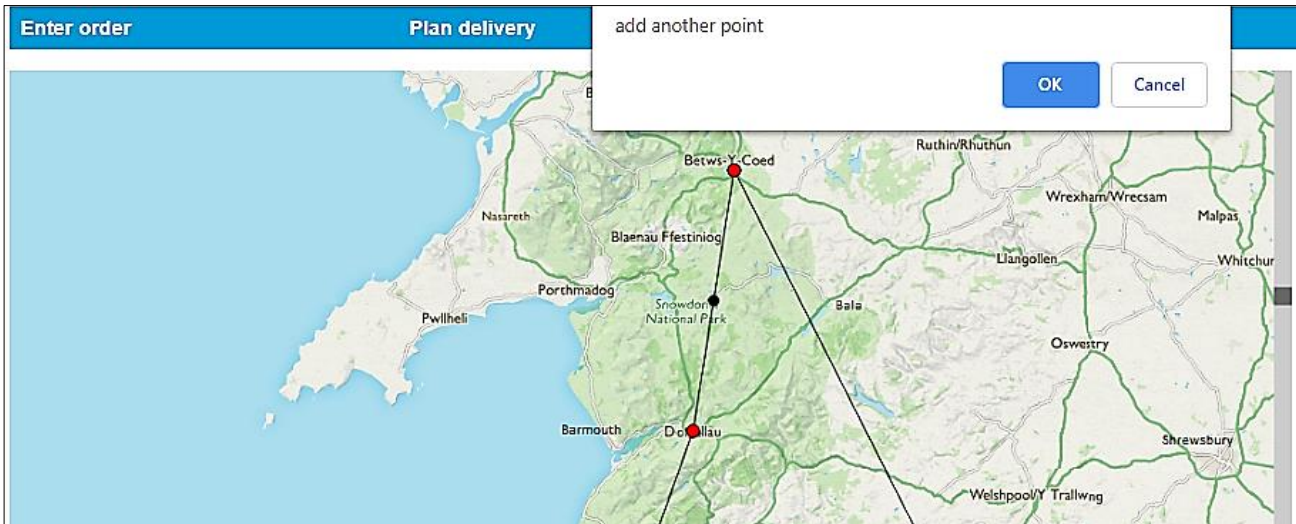
var scrollPosition=<? echo $scrollPosition ?>;

var vPos;
var ratio;
var scrollSelected = false;

```

These changes were to ensure that the map is displayed in the same scroll position when the page is reloaded.

Save **route.php** and copy it to the server. Refresh the page, then move the mouse pointer to the mid point of a line segment, so that a circular marker appears. Click the mouse on the marker. The confirm dialogue should appear.



Click 'Cancel' and check that the map is reloaded in the same scroll position, but with the selected point removed. We can now move on to add a point to the linked list.

Open a blank file and add the lines of program code below. Save the file as **addPoint.php** and copy it to the server.

This page will not be visible to the user when the program runs. It is loaded when the 'OK' button is clicked to add a route point, then begins by collecting information about the line segment which has been selected: the **roadPointID** values for the points at each end of the segment, and the map coordinates of the mid point.

```

<?
    $scrollTop=$_REQUEST['scrollTop'];
    $current=$_REQUEST['current'];
    $next=$_REQUEST['next'];
    $midX=$_REQUEST['midX'];
    $midY=$_REQUEST['midY'];
    include('RoadPoints.php');
    RoadPoints::insertPoint($current,$next,$midX,$midY);
    header('Location: route.php?scrollTop='.$scrollTop);
?>

```

The program then runs a method in the **RoadPoints** class file which will insert the new point into the linked list. We will add this method now.

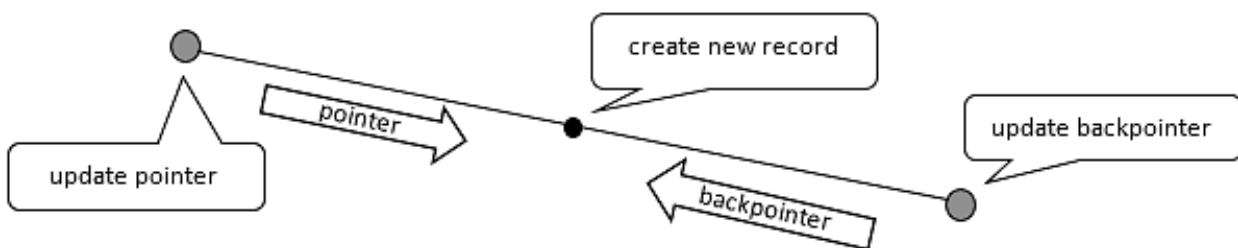
Open the **RoadPoints.php** file and add the **insertPoint()** method shown below.


```

public static function insertPoint($current,$next,$midX,$midY)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="INSERT INTO roadPoints VALUES ('','0','0','$midX',
                                           '$midY','$next','$current')";
    $result=mysqli_query($conn, $query);
    $newRoadPointID = mysqli_insert_id($conn);
    $query="UPDATE roadPoints SET pointer='$newRoadPointID'
           WHERE roadPointID = '$current'";
    $result=mysqli_query($conn, $query);
    $query="UPDATE roadPoints SET backpointer='$newRoadPointID'
           WHERE roadPointID = '$next'";
    $result=mysqli_query($conn, $query);
    mysqli_close($conn);
}
}
?>

```

This method creates a new **roadPoint** record in the database, with the **pointer** and **backpointer** values set to the **roadPointID** values at the two ends of the original line segment. The end pointers are then updated to point to the new intermediate point.



Save the **RoadPoints.php** file and copy it to the server.

Return to the **route.php** file and locate the **plotLink()** function. Add lines of code as shown below to mark the position of any intermediate points between the delivery customers.

```

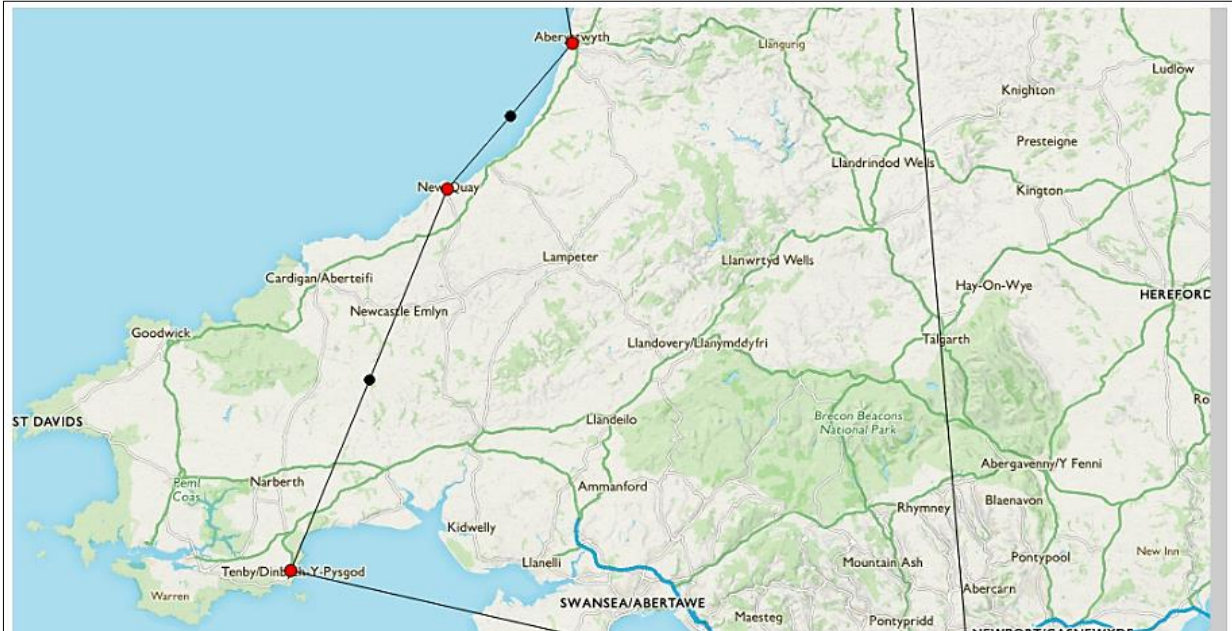
while (finished==false)
{
    x1=roadPoints[current].xpos;
    y1=roadPoints[current].ypos;
    next=roadPoints[current].pointer;
    x2=roadPoints[next].xpos;
    y2=roadPoints[next].ypos;
    line(x1,y1,x2,y2);

    fill(0);
    ellipse(x1,y1,8,8);
    ellipse(x2,y2,8,8);

    x=mouseX;
    y=mouseY;
    y=y+tv;
}

```

Save **route.php** and copy it to the server. Refresh the page, then click the mouse on the mid point of a line segment. Select 'OK' in the confirm dialogue box. The page should refresh at the same scroll position with the additional point now added to the line segment. Repeat the process to insert several more points between delivery locations.



Go to the PHP MyAdmin website and examine the roadPoints table. Records will have been added for the intermediate points between delivery locations, and are connected back to the original records by means of the forwards and backwards pointers, as in this example. You may need to set the database table to display more than the default number of rows of data.

roadPointID					pointer	backpointer
12	0	0	362	49	-1	11
13	19	4	362	49	22	-1
14	0	0	461	647	-1	22
15	13	6	229	1081	16	-1
16	0	0	791	1209	-1	15
17	6	0	791	1209	18	-1
18	0	0	738	585	-1	17
19	0	19	738	585	20	-1
20	0	0	362	49	-1	19
21	0	0	410	707	2	1
22	0	0	412	348	14	13
23	0	0	294	924	4	3

We will now allow the user to move the inserted points so that they lie accurately on the road network. Return to the **route.php** file and insert a **movePoint()** function underneath the **draw()** function. Add a line of program code to **draw()** to call this function, as shown below.


```

        if (scrollPosition>640)
        {
            scrollPosition=640;
        }
    }
}
drawRoute();
movePoint();
}

function movePoint()
{
    x=mouseX;
    if (x<=960)
    {
        y=mouseY;
        y=y+tv;
        if (mouseIsPressed==true)
        {
            if (dragging==false)
            {
                dragging=true;
            }
        }
        if (mouseIsPressed==false)
        {
            if (dragging==true)
            {
                dragging=false;
            }
        }
        for (i=1;i<=pointCount; i++)
        {
            xpos=roadPoints[i].xpos;
            ypos=roadPoints[i].ypos;
            if ((Math.abs(x-xpos)<10)&&(Math.abs(y-ypos)<10))
            {
                if ((roadPoints[i].pointer>-1)&&(roadPoints[i].backpointer>-1))
                {
                    if (dragging==true)
                    {
                        roadPoints[i].xpos = x;
                        roadPoints[i].ypos = y;
                        selected=i;
                    }
                }
            }
        }
    }
}

function drawRoute()
{
    for (var i=0; i<(locationTotal-1); i++)
    {

```

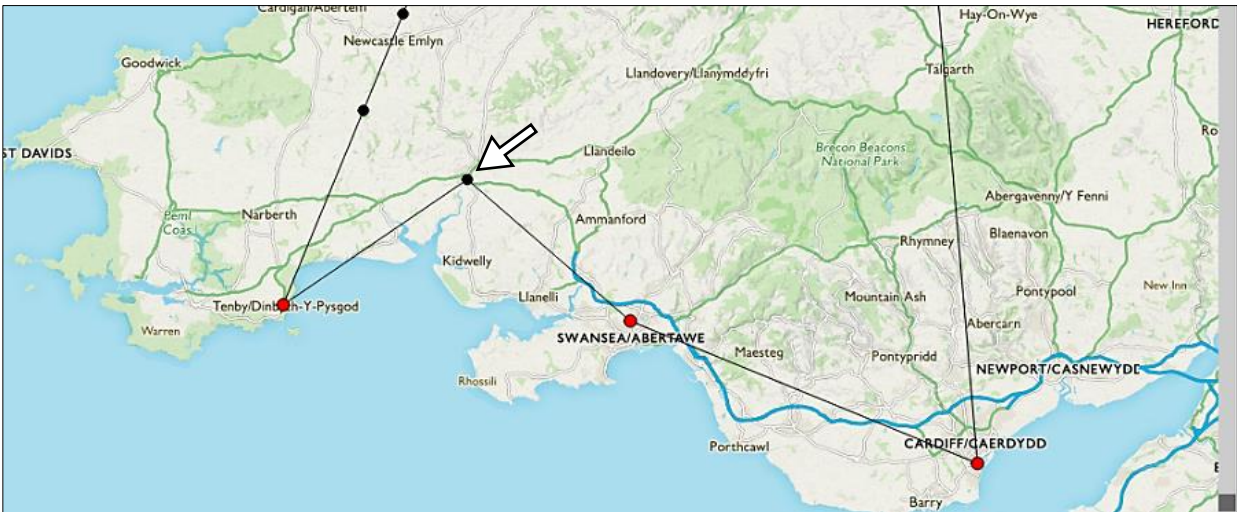
The function begins by finding the **x** and **y** coordinates of the mouse. A variable '**dragging**' is set to '**true**' if the mouse is being moved with the button held down. Each of the intermediate route points is then

checked in turn. If the mouse lies close to a point and is being dragged, the point moves to the current mouse position. The roadPointID of this point is recorded as the variable 'selected'.

The two additional variables **dragging** and **selected** should be defined as global, so that they can be used by several functions. Insert these variables near the start of the **<script>** block as shown below.

```
var vPos;  
var ratio;  
var scrollSelected = false;  
  
var dragging=false;  
var selected=0;  
  
function preload()  
{  
  img1=loadImage("map.png");  
}
```

Save the **route.php** file and copy it to the server. Refresh the page, then test the **movePoint()** function by holding down the mouse on an intermediate route point and dragging the mouse to another position. The point should follow. Check that the delivery locations marked by red circles cannot be moved.



The next step is to allow the new point location to be updated in the database table, so that the point remains in the required position when the page is reloaded.

Return to the **movePoint()** function in the **route.php** file. Add lines of program code as shown below. If the mouse button is released while a route point is being dragged, a new function **loadPage2()** will be called. This will display a confirm dialogue box, and the user must click 'OK' before changes are applied to the database record.

```
if (mouseIsPressed==false)  
{  
  if (dragging==true)  
  {  
    dragging=false;  
  
    if (selected>0)  
      loadPage2();  
  }  
}  
for (i=1;i<=pointCount; i++)
```

Add the **loadPage2()** function immediately below the **movePoint()** function.

```
function movePoint()
{
    . . . . .
}

function loadPage2()
{
    choice = confirm('change position of point');
    if (choice==true)
    {
        window.location = "movePoint.php?scrollPosition="+scrollPosition+
            "&selected="+selected+"&x="+int(x)+"&y="+int(y);
    }
    else
        window.location = "route.php?scrollPosition="+scrollPosition;
}

function drawRoute()
{
    for (var i=0; i<(locationTotal-1); i++)
    {
```

Save the **route.php** file and copy it to the server.

If the user confirms to save the new position of the route point, another page will be loaded. We will create this next. Open a blank file and add the lines of program code below. Save the file as **movePoint.php** and copy it to the server.

```
<?
    $scrollPosition=$_REQUEST['scrollPosition'];
    $selected=$_REQUEST['selected'];
    $x=$_REQUEST['x'];
    $y=$_REQUEST['y'];
    include('RoadPoints.php');
    RoadPoints::movePoint($selected,$x,$y);
    header('Location: route.php?scrollPosition='.$scrollPosition);
?>
```

The **movePoint.php** page will not be visible to the user when the program is running. The page collects the **routePointID** of the moved point, along with the x and y coordinates of its new position. This information is passed to a **movePoint()** method in the **RoadPoints** class, which will update the corresponding database record.

Go to the **RoadPoints.php** file and add the **movePoint()** method as shown below. Save the **RoadPoints.php** file and copy it to the server.

```
public static function movePoint($selected,$x,$y)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="UPDATE roadPoints SET xpos='$x', ypos='$y'
        WHERE roadPointID='$selected'";
    $result=mysqli_query($conn, $query);
    mysqli_close($conn);
}

?>
```

Return to **route.php**. To avoid a conflict occurring between the functions for adding and moving a point, we will make a slight modification to the **plotLink()** function. Change the line shown below.

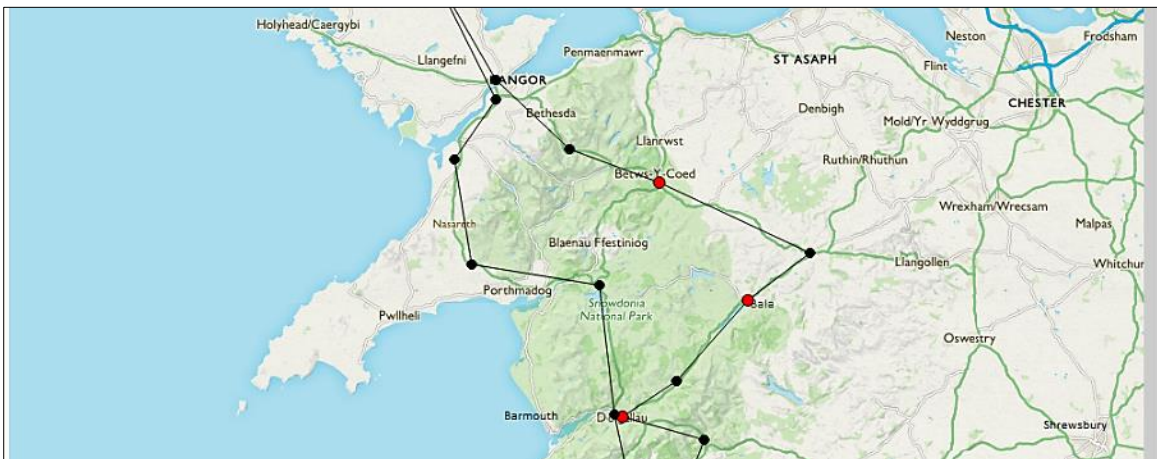
```

        if ((Math.abs(x-midX)<10)&&(Math.abs(y-midY)<10))
        {
            fill(0);
            ellipse(midX,midY,8,8);
            fill(255,0,0);

            if ((mouseIsPressed==true)&&(dragging==false))
            {
                loadPage();
            }
        }
    }

```

Save **route.php** and copy it to the server. Refresh the page. It should now be possible to add new intermediate point between any pair of existing points, then drag the new point to its correct position on the map. The position will then be updated in the database table. This can be repeated as necessary to show the route with sufficient accuracy. The route mileage will be calculated according to the sequence of links shown on the map.



If at any time you wish to clear the intermediate points added between customers and begin again with a clear map, this can be done by going to the PHP database and deleting all data from the **roadPoints** table. The program will then automatically recreate direct links between each pair of customers.

This completes the map display. We will now add a web page to display the sequence of delivery locations and distances. Before creating the new page, return to the **route.php** file and add lines of program code to display a button with the caption 'Display delivery addresses', as shown below. Save the **route.php** file and copy it to the server.

```

        RoadPoints::createLink($start,$finish,$x[$i],$y[$i],$x[$i+1],$y[$i+1]);
    }
}
$pointCount = RoadPoints::loadRoadPoints();
$roadPointsJSON = json_encode(RoadPoints::$location);
?>
<p>
    <form method=post action='deliveryList.php'>
        <input type=submit value='Display delivery addresses'>
    </form>
</p>
</script>

```

Refresh the page and check that the button is displayed.

Enter order	Plan delivery
<input type="button" value="Display delivery addresses"/>	

We will now create the **deliveryList** page. Open a blank file and add the lines of program code below.

```
<?
    session_start();
?>
<html>
<head>
    <title> Delivery route </title>
    <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
    <?
        include('staffMenu.php');
    ?>
    <p>
    <form method=post action='route.php'>
    <input type=submit value='Display delivery map'>
    </form>
    <table class=stock cellspacing=10px>
    <tr class=stock>
        <th class=stock>CustomerID</th>
        <th class=stock>Company</th>
        <th class=stock>Town</th>
        <th class=stock>Address</th>
        <th class=stock>Miles</th>
        <th class=stock>Journey total</th>
    </tr>
    </table>
</body>
</html>
```

Save the file as **deliveryList.php** and copy it to the server.

Run the website, select a series of orders for delivery and continue to the delivery map. Click the 'Display delivery addresses' button. A new page should open with a series of table headings displayed. This will list the customers in the order of delivery, along with the mileage between delivery points.

Enter order	Plan delivery	Add customer			
<input type="button" value="Display delivery map"/>					
CustomerID	Company	Town	Address	Miles	Journey total

Return to the **deliveryList.php** file and add the block of PHP code shown below. This accesses the **deliveryCustomer** table in the database and obtains the company names and addresses for each of the delivery points. These will be listed in the order produced by the Nearest Neighbour algorithm, and may begin and end at a location other than the Newtown depot.

```

<body>
  <?
  include('staffMenu.php');

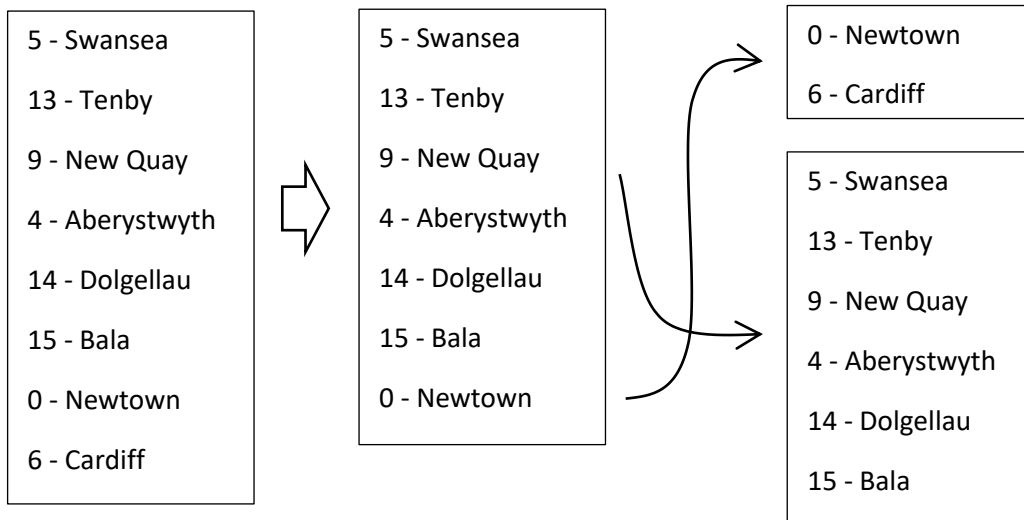
  include('DeliveryCustomer.php');
  include('RoadPoints.php');
  RoadPoints::loadRoadPoints();
  $custID = $_SESSION["routeCustomerWanted"];
  $locationTotal=count($custID);
  for ($i=0; $i<$locationTotal; $i++)
  {
    $customerIDwanted = $custID[$i];
    if ($customerIDwanted==0)
    {
      $town[$i]='Newtown';
    }
    else
    {
      DeliveryCustomer::loadCustomerByID($customerIDwanted);
      $town[$i]=DeliveryCustomer::$customerObj[1]->getTown();
      $companyName[$i]=DeliveryCustomer::$customerObj[1]->getCompanyName();
      $address[$i]=DeliveryCustomer::$customerObj[1]->getAddress();
    }
  }

  ?>
  <p>
  <form method=post action='route.php'>
  <input type=submit value='Display delivery map'>
  </form>
  <table class=stock cellpadding=10px>
    <tr class=stock>
      <th class=stock>CustomerID</th>
      <th class=stock>Company</th>

```

The delivery sequence which we will display on the web page should represent the actual delivery journey beginning and ending at the Newtown depot. Some reorganisation is therefore necessary:

- The sequence produced by the algorithm will duplicate a location at both the start and end of the list: in this case **Swansea**. The entry at the end of the list is first removed.
- A new list is created, beginning with the entry for Newtown and adding any entries which follow this: in this case **Cardiff**.
- The first part of the original list is then added, copying the entries up to and including **Newtown**.



Return to the **deliveryList.php** file and add the block of program code shown below.

```

        <th class=stock>Address</th>
        <th class=stock>Miles</th>
        <th class=stock>Journey total</th>
    </tr>
    <?
        $locationTotal--;
        for ($i=0; $i<$locationTotal;$i++)
        {
            if ($custID[$i]==0)
            {
                $start=$i;
            }
        }
    ?>
</table>
</body>

```

The purpose of this section of program is to determine the position in the delivery sequence of the Newtown depot (customer ID = 0). This information is then used to split the original list at the correct point.

Continue by adding the program code shown below. This will begin to display the last section of the original delivery list, starting with an entry for the Newtown depot.

```

        for ($i=0; $i<$locationTotal;$i++)
        {
            if ($custID[$i]==0)
            {
                $start=$i;
            }
        }

        $total=0;
        for ($i=$start; $i<$locationTotal; $i++)
        {
            echo"<tr class=stock>";
            echo"<td class=stock>$custID[$i]";
            if ($custID[$i]==0)
            {
                echo"<td class=stock>Start from Newtown Depot</td><td class=stock>
                </td><td class=stock><td class=stock></td><td class=stock>0</td>";
            }
            echo"</tr>";
        }

    ?>
</table>
</body>
</html>

```

Continue by adding the program code shown below. This will display the last section of the original delivery list, from Newtown onwards. After adding each company name, town and address to the table, the program will call a method in the **RoadPoints** class to calculate the mileage from the previous delivery

point. To complete the route, the program adds the first group of delivery locations from the original list, up to and including the Newtown depot where the journey ends. Save the **deliveryList.php** file and copy it to the server.

```

        if ($custID[$i]==0)
        {
            echo"<td class=stock>Start from Newtown Depot</td><td class=stock>
            </td><td class=stock><td class=stock>0</td><td class=stock>0</td>";
        }

        else
        {
            echo"<td class=stock>$companyName[$i] </td>";
            echo"<td class=stock>$town[$i] </td>";
            echo"<td class=stock>$address[$i] </td>";
            $previous=$i-1;
            $distance = RoadPoints::
                calculateDistance($custID[$previous],$custID[$i]);
            echo"<td class=stock>$distance</td>";
            $total = $total+$distance;
            echo"<td class=stock>$total</td>";
        }
        echo"</tr>";
    }
    for ($i=0; $i<= $start;$i++)
    {
        echo"<tr class=stock>";
        echo"<td class=stock>$custID[$i] </td>";
        if ($custID[$i]==0)
        {
            echo"<td class=stock>Finish at Newtown Depot</td>
            <td class=stock><td class=stock></td>";
            $previous=$i-1;
            if ($previous<0)
                $previous=$locationTotal-1;
            $distance = RoadPoints::
                calculateDistance($custID[$previous],$custID[$i]);
            echo"<td class=stock>$distance</td>";
            $total = $total+$distance;
            echo"<td class=stock>$total</td>";
        }
        else
        {
            echo"<td class=stock>$companyName[$i] </td>";
            echo"<td class=stock>$town[$i] </td>";
            echo"<td class=stock>$address[$i] </td>";
            if ($i==0)
                $previous=$locationTotal-1;
            else
                $previous=$i-1;
            $distance = RoadPoints::
                calculateDistance($custID[$previous],$custID[$i]);
            echo"<td class=stock>$distance</td>";
            $total = $total+$distance;
            echo"<td class=stock>$total</td>";
        }
        echo"</tr>";
    }
    ?>
</table>

```


We just need to add a method to the RoadPoints class to calculate the distances between delivery points. Re-open **RoadPoints.php** and add the method shown below. This finds the starting point for the linked list of points marking the route between the two delivery points. Each pair of points along the linked list are then considered in turn, and the distance between them in screen pixels is calculated using Pythagoras' theorem. When all points have been processed, the total distance is converted from screen pixels to miles according to the scale of the map on the screen.

```

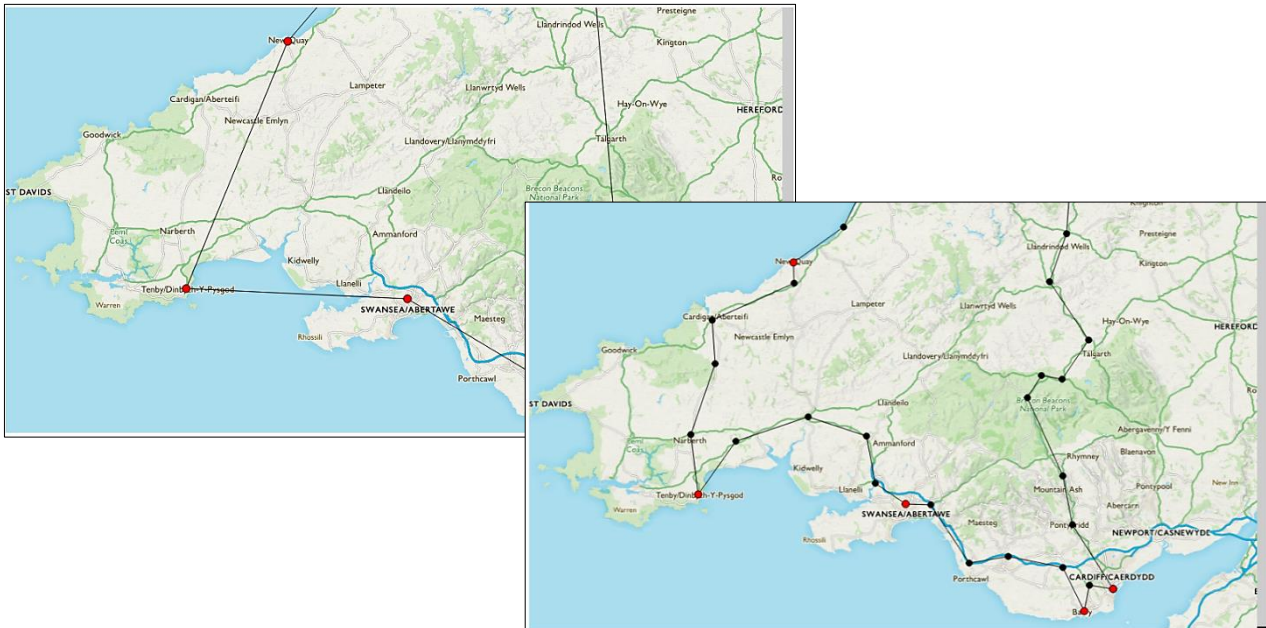
public static function calculateDistance($fromCustomer,$toCustomer)
{
    $result=0;
    $pointCount=RoadPoints::loadRoadPoints();
    for ($i=1;$i<=$pointCount;$i++)
    {
        $a = RoadPoints::$location[$i]->fromCustomer;
        $b = RoadPoints::$location[$i]->toCustomer;
        if (((($fromCustomer==$a)&&($toCustomer==$b)) ||
            (($fromCustomer==$b)&&($toCustomer==$a)))
        {
            $current=$i;
        }
    }
    $finished=false;
    $total=0;
    while($finished==false)
    {
        $xCurrent=intval(RoadPoints::$location[$current]->xpos);
        $yCurrent=intval(RoadPoints::$location[$current]->ypos);
        $next=RoadPoints::$location[$current]->pointer;
        $xNext=intval(RoadPoints::$location[$next]->xpos);
        $yNext=intval(RoadPoints::$location[$next]->ypos);
        $xTerm = $xCurrent-$xNext;
        $yTerm = $yCurrent-$yNext;
        $distance = sqrt($xTerm*$xTerm + $yTerm*$yTerm);
        $total = $total + $distance;
        if (RoadPoints::$location[$next]->pointer == -1)
            $finished=true;
        $current=$next;
        $count++;
    }
    $total = $total / 8.3;
    return intval($total);
    return $total;
}
}
?>

```

Save the **RoadPoints.php** file and copy it to the server. Refresh the **deliveryList** page and check that the sequence of customers along the delivery route is displayed correctly, as in this example.

CustomerID	Company	Town	Address	Miles	Journey total
0	Start from Newtown Depot			0	0
14	Cader Idris Climbing Shop	Dolgellau	Bridge Street	36	36
15	Lakeside Watersports	Bala	Tegid Avenue	18	54
7	Snowdonia Mountainsport	Betws-y-Coed	Ffordd Gethin	25	79
19	Anglesey coast and mountain centre	Amlwch	Heol y Bont	35	114
4	Cambrian Outdoors	Aberystwyth	Marine Terrace	94	208
9	Sailing Centre	New Quay	Harbour Road	19	227
13	Pembrokeshire Watersports Centre	Tenby	Harbour Terrace	47	274
6	Bay Watersports	Cardiff	Marine Drive	85	359
0	Finish at Newtown Depot			85	444

This completes the program output. However, one slight problem remains. When we set up the Nearest Neighbour Algorithm calculation, we used straight line distances between pairs of delivery points rather than the more accurate linked list distances following the road network. The accuracy of the program can be improved by using the linked list distances.



Reopen the **planRoute.php** file. Locate the block of PHP code which creates the table of distances between delivery locations. Remove this block, so that it can be replaced by a new table in which distances are obtained from the linked lists of points.

```

        $count++;
    }
}
echo"<p>";

echo"<table class=stock>";
echo"<tr>";
echo"<td class=stock>";
for($i=0;$i<=$routeCount;$i++)
{
    echo"<td class=stock>".$routeCustomerID[$i]." ".$routeTown[$i]."/td>";
}
for($i=0;$i<=$routeCount;$i++)
{
    echo"<tr>";
    echo"<td class=stock>".$routeCustomerID[$i]." ".$routeTown[$i]."/td>";
    for ($j=0;$j<=$routeCount;$j++)
    {
        $Xdifference = $routeX[$i]-$routeX[$j];
        $Ydifference = $routeY[$i]-$routeY[$j];
        $distance = $Xdifference*$Xdifference+$Ydifference*$Ydifference;
        $distance = sqrt($distance);
        echo"<td class=stock>".number_format($distance,0);
        $link[$i][$j]=$distance;
    }
}
echo"</table>";

for ($start=0;$start<=$routeCount;$start++)

```

REMOVE <TABLE> BLOCK

Replace the original table with the block of code shown below. This begins by loading the set of RoadPoint objects from the database table. Loops again access each element of the table which represent connections between pairs of delivery locations. A check is made to see whether a linked list of points has been created for the current connection; if so, this is used to calculate the distance. If not, a straight line distance is calculated by Pythagoras' formula as in the original table.

```

    $count++;
  }
}
echo"<p>";

include('RoadPoints.php');
$pointCount=RoadPoints::loadRoadPoints();
echo"<table class=stock>";
echo"<tr>";
echo"<td class=stock>";
for($i=0;$i<=$routeCount;$i++)
{
  echo"<td class=stock>".$routeCustomerID[$i]."  ".$routeTown[$i]."</td>";
}
for($i=0;$i<=$routeCount;$i++)
{
  echo"<tr>";
  echo"<td class=stock>".$routeCustomerID[$i]."  ".$routeTown[$i]."</td>";
  for ($j=0;$j<=$routeCount;$j++)
  {
    $a = $routeCustomerID[$i];
    $b = $routeCustomerID[$j];
    $found=false;
    for ($n=1;$n<=$pointCount;$n++)
    {
      $fromCustomer = RoadPoints::$location[$n]->fromCustomer;
      $toCustomer = RoadPoints::$location[$n]->toCustomer;
      if (((($fromCustomer==$a)&&($toCustomer==$b))||
          (($fromCustomer==$b)&&($toCustomer==$a))))
      {
        $found=true;
      }
    }
    if (($a==0)&&($b==0))
      $found=false;
    if ($found==true)
    {
      $distance = RoadPoints::calculateDistance($a,$b);
    }
    else
    {
      $Xdifference = $routeX[$i]-$routeX[$j];
      $Ydifference = $routeY[$i]-$routeY[$j];
      $distance = $Xdifference*$Xdifference+$Ydifference*$Ydifference;
      $distance = sqrt($distance);
      $distance = $distance/ 8.3;
    }
    echo"<td class=stock>".number_format($distance,0);
    $link[$i][$j]=$distance;
  }
}
echo"</table>";

for ($start=0;$start<=$routeCount;$start++)

```

Save **planRoute.php** and copy it to the server. Run the web site and select a set of orders for delivery. Continue to the planRoute page, where the updated table of distances will be displayed. The distances are now shown in miles, rather than screen pixels, and have been converted according to the scale of the map image on the screen.

	0 Newtown	6 Cardiff	14 Dolgellau	16 Pwllheli	9 New Quay	5 Swansea	19 Amlwch	15 Bala	32 Barry
0 Newtown	0	86	34	68	54	79	99	53	78
6 Cardiff	86	0	97	117	114	41	149	106	8
14 Dolgellau	34	97	0	35	44	78	64	18	99
16 Pwllheli	68	117	35	0	81	91	41	54	149
9 New Quay	54	114	44	81	0	46	110	61	85
5 Swansea	79	41	78	91	46	0	127	92	38
19 Amlwch	99	149	64	41	110	127	0	62	150
15 Bala	53	106	18	54	61	92	62	0	109
32 Barry	78	8	99	149	85	38	150	109	0

current = 0 - Newtown - REMAINING: Cardiff(86) Dolgellau(34) Pwllheli(68) New Quay(54) Swansea(79) Amlwch(99) Bala(53) Barry(78)
current = 14 - Dolgellau - REMAINING: Cardiff(97) Pwllheli(35) New Quay(44) Swansea(78) Amlwch(64) Bala(18) Barry(99)
current = 15 - Bala - REMAINING: Cardiff(106) Pwllheli(54) New Quay(61) Swansea(92) Amlwch(62) Barry(109)
current = 16 - Pwllheli - REMAINING: Cardiff(117) New Quay(81) Swansea(91) Amlwch(41) Barry(149)
current = 19 - Amlwch - REMAINING: Cardiff(149) New Quay(110) Swansea(127) Barry(150)
current = 9 - New Quay - REMAINING: Cardiff(114) Swansea(46) Barry(85)
current = 5 - Swansea - REMAINING: Cardiff(41) Barry(38)
current = 32 - Barry - REMAINING: Cardiff(8)
current = 6 - Cardiff
Return distance to Newtown = 86
TOTAL DISTANCE = 435

Click the 'continue' button and the route map should be displayed as before.

Further development

The delivery planning system developed in this example project contains the basic elements required to operate the system, but further editing and updating facilities could be added. For example, there might be an option to upload maps of other areas, with the user entering information about the map scale so that distances could be calculated.

Other route planning systems might be developed, for example: for home parcel deliveries, or for an engineer who has to visit a number of premises to check gas or electrical installations.

Summary of the object structures

Staff

A Staff object contains the staffID which is set as an auto-number, along with the user name and password. The public method **checkPassword()** calls the private method **checkUser()** to examine each Staff object in turn, then returns an overall true/false result depending on whether valid log-in details were found.

Staff
- staffID: integer - userName: string - password: string
+ constructor(userName, password) - checkUser(userName, password): boolean + <u>checkPassword(userName, password): boolean</u>

OutdoorEquipment

Objects in this class represent the products distributed by the company. Attributes include: the stockcode and item title, a written description and picture, and the price. Methods are provided to add new stock records, load all stock items for display, or select a particular stock item by its ID number. Attributes are accessed by means of a set of **get()** methods.

DeliveryCustomer

Objects represent the customers who receive deliveries from the company. In addition to name and address, the attributes include the map coordinates for the delivery location. Methods are provided to add new customers, load all customers for listing on a page, or to select a particular customer by ID number. Attributes are accessed by means of a set of **get()** methods.

OutdoorEquipmentOrder

Objects represent orders received from customers. Attributes identify the customer by ID number, and provide information on whether the order is awaiting delivery or has been delivered. Methods are provided to add an order, load all order for display on screen, and to record that an order has been delivered. Attributes are accessed by means of a set of **get()** methods.

EquipmentOrderItem

Objects represent individual item entries on a customer order. Attributes identify the order and product by their ID numbers, and specify the quantity of the product ordered. Methods are provided to add an item to an order and to load all order items for a particular order, identified by its orderID number. Attributes are accessed by means of a set of **get()** methods.

RoadPoint

Objects represent points along the delivery route. Attributes specify the map coordinates of the point and indicate whether it represents a customer location or an intermediate point on the connecting road network. Pointers create linked lists of road points between delivery points. A method is provided to check whether a linked list of points already exists between two customers, and a method can create a simple straight line connection where none exists. Methods load all road points, and can insert a new point midway between two existing road points. The **movePoint()** method allows a point to be dragged by mouse pointer to a new location on the map. The **calculateDistance()** method finds the distance in miles between two specified road points.

