

5

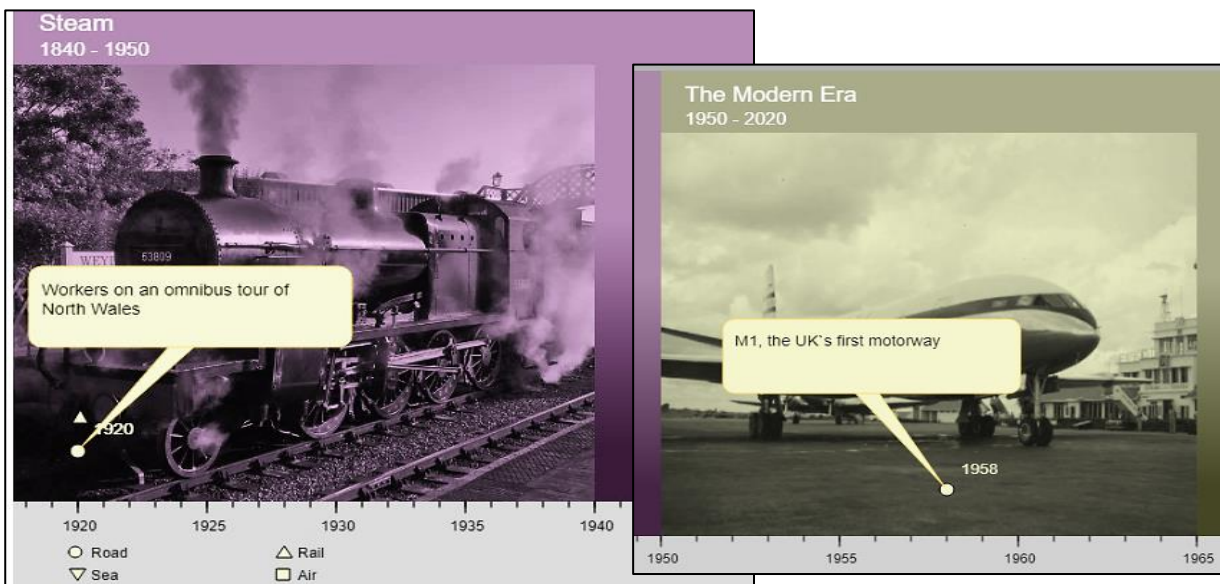
Historical timeline

Scenario

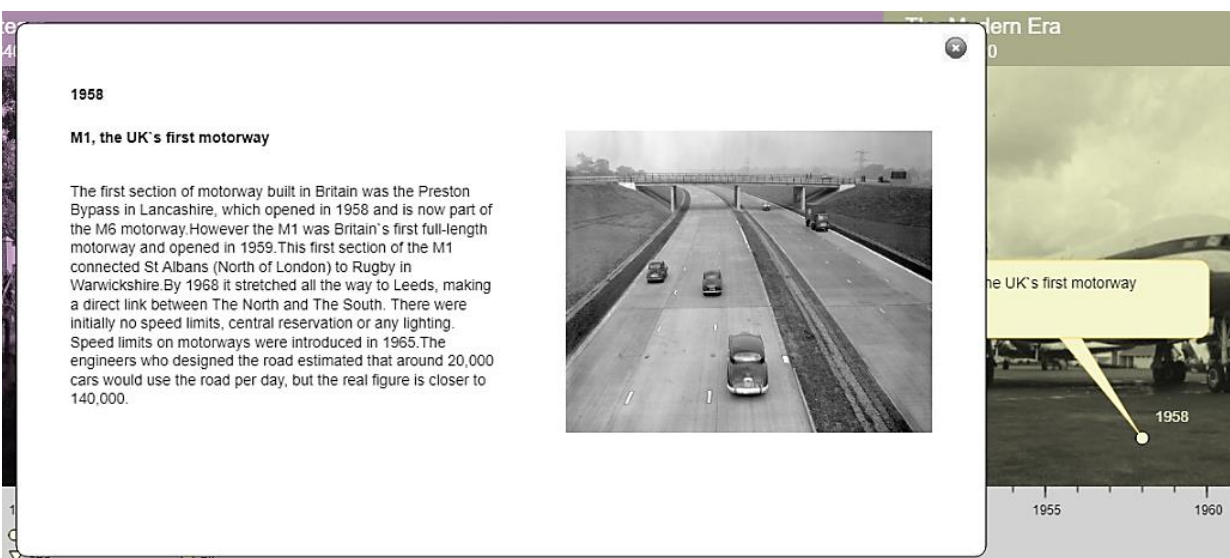
A history society wishes to set up an on-line application to create historical timelines for display on the internet. Example themes may be: the history of a particular city or region, or the historical development of a particular technology such as 'Domestic Housing in Britain'. A timeline should scroll to show an extended number of years, divided into periods such as 'the Victorian Era' or 'the post-War years'. Symbols on the timeline will link to information panels for events occurring in particular years.

Registered users should have a facility to add information about historical events. The uploaded entries are to be approved by staff before display on the publicly available timeline web page. Staff may also create timelines on new themes and add these to the web site.

It will be possible to specify themes which cut across time periods. These themes will be listed in a key, and appear on the timeline by different symbols. For example, in a History of Transport timeline we might choose the themes: **Road**, **Sea**, **Rail**, and **Air** as different forms of transport.

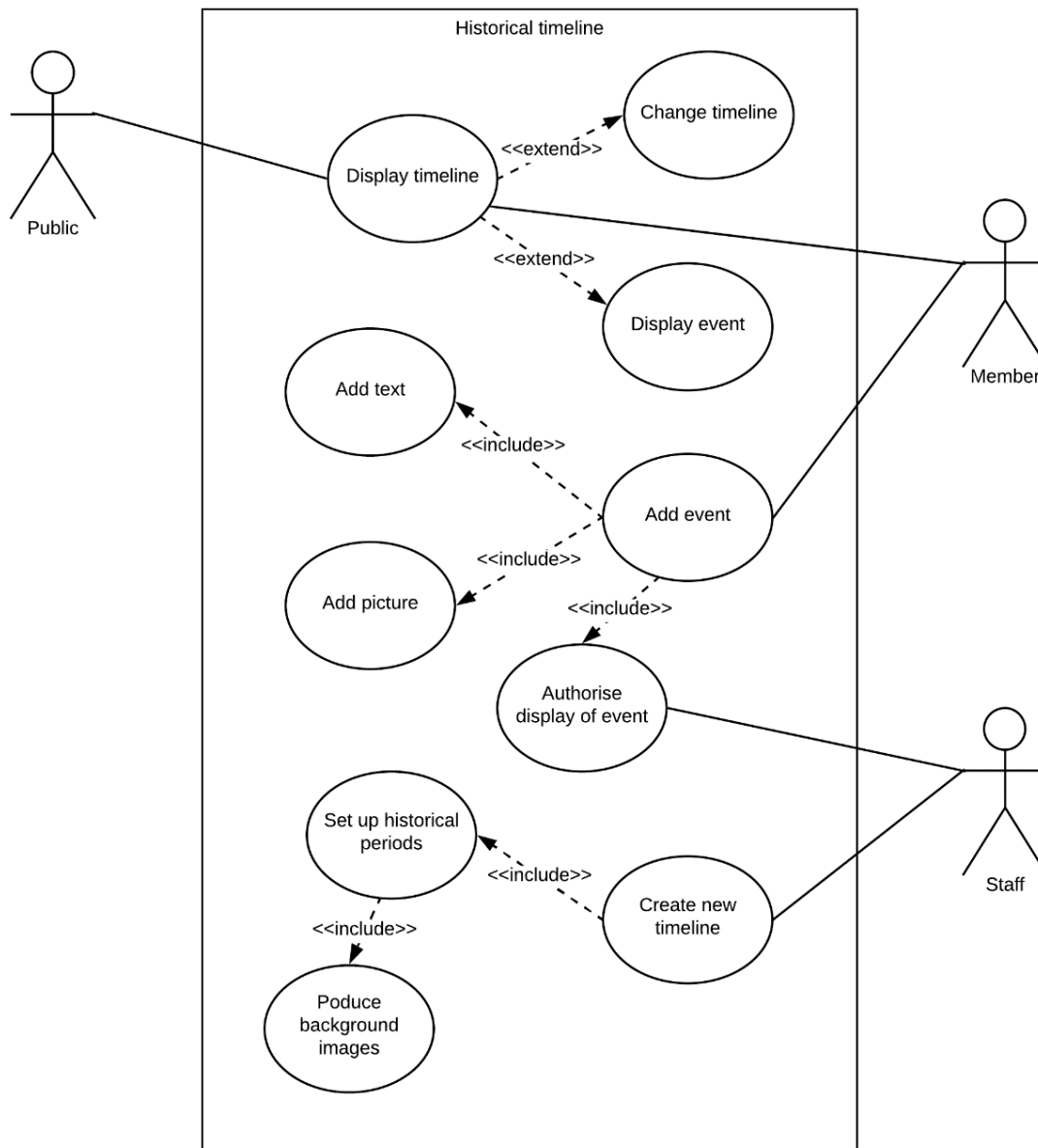


If the mouse is clicked on a symbol displayed on the timeline, an information panel will be opened.



Design

The objectives of the project can be summarised in a use case diagram.



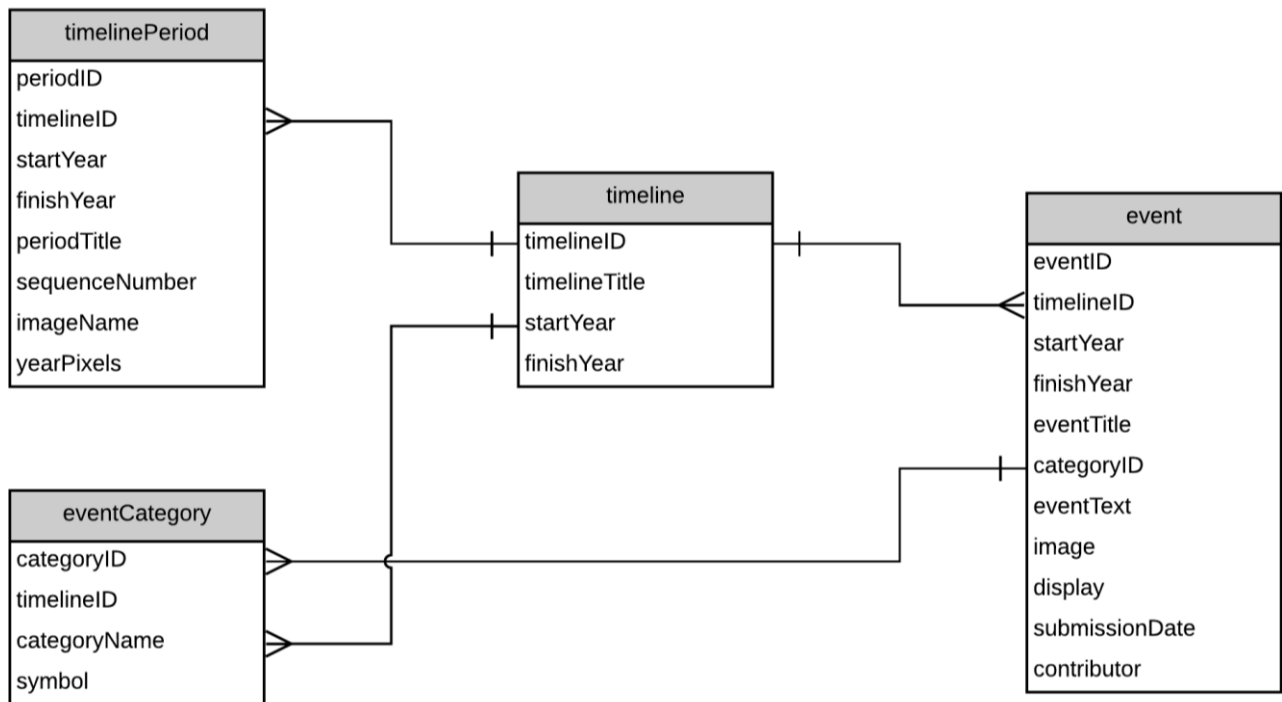
Three types of actor will interact with the website:

Public users will go to the site to view the historical timelines. They may select information on events of interest, or may choose timelines from the set available.

Registered **members** may view the timelines, and may additionally add historical events. Text and a picture image will be required for each event.

Staff of the historical society have an overview of the website. Staff will approve articles submitted by members before they are displayed on the timeline. Staff may also create new timelines by setting the start and finish years, and dividing the timeline into a series of historical periods. The website will provide graphics tools to produce background images for the historical periods.

The operation of the website will be centred around a series of database tables.



The **timeline** table contains the title and year range for each timeline.

The **timelinePeriod** table defines the sequence of historical periods within the timeline, recording the start and finish dates and the background image to be displayed. The **yearPixels** value determines the horizontal year scale to be used when displaying the period.

The **event** table records historical events which are linked to particular years, and includes text and the file names of images for display.

The **eventCategory** table records the classifications used for events within a particular timeline. Different symbols will then identify events within the specified categories, such as events related to politics or industry.

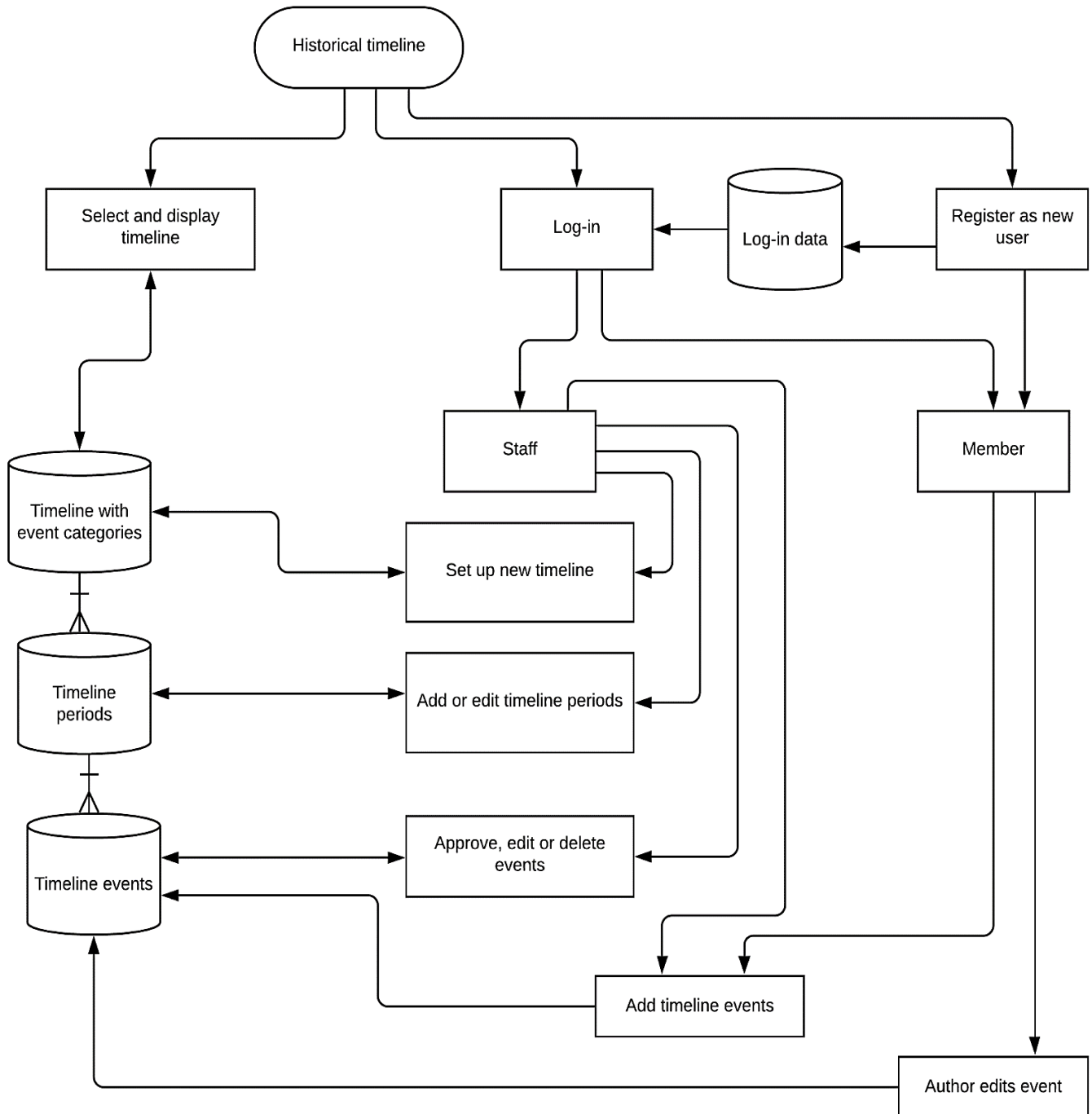
Links between the web page functions and the database tables are illustrated in the flowchart shown on the next page.

Programming techniques

The web site will be created using a mixture of **PHP**, **Javascript** and **p5.js** programming languages. Text content for the site will be stored in an SQL database, then used to create objects in PHP for access by the web pages. Scrolling and interactive graphics will be created with the Javascript-based **p5.js** graphics language, with native Javascript code forming links between PHP and the screen graphics display.

Method

We will begin by producing a log-in system to allow access to the website functions for members and staff of the society. A historical timeline will then be created and tested. Programming will follow the sequence outlined in the flowchart above: a new timeline will be defined; historical periods will be set up, including the creation of background images; then information about historical events will be added.



A password protected log-in system is needed for both staff and members of the historical society. It is simplest to use a single database table to hold all usernames and passwords, and to include a field which identifies the user's status as 'staff' or 'member'. E-mail addresses will also be held in the table, to allow easy communication between staff and members.

Go to the PHP MyAdmin website for the database and select the option to add a new table. Create a table with the name **timelineLogin** and add the fields shown below. The integer **loginID** field should be specified as the primary key, and set to auto-increment as records are added. Other fields are of the data type **varchar** with the sizes shown below. The procedure for setting up a MySQL database table was described in more detail previously in Chapter 2: Hardware Store.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	loginID	int(11)			No	None		AUTO_INCREMENT
2	forename	varchar(30)	latin1_swedish_ci		No	None		
3	surname	varchar(30)	latin1_swedish_ci		No	None		
4	email	varchar(30)	latin1_swedish_ci		No	None		
5	userName	varchar(20)	latin1_swedish_ci		No	None		
6	password	varchar(20)	latin1_swedish_ci		No	None		
7	status	varchar(12)	latin1_swedish_ci		No	None		

Add several records to the **timelineLogin** table. Include both 'staff' and 'member' entries in the status field.

loginID	forename	surname	email	userName	password	status
1	Dafydd	Jones	jones1@historygroup.org	Jones1	abc	staff
2	Robert	Brown	brown1@historygroup.org	Brown1	xyz	staff
3	Angela	Pritchard	angelap47@telecom.com	angela47	pqr	member

Set up folders with the name **'timeline'** on the local computer and on the server to hold the project files. We will now create a log-in page for the web site. Open a blank file and add the lines of program code shown in the two boxes below.

```
<?
    session_start();
    $_SESSION['login']="";
?>
<html>
<head>
<title>Historical timeline</title>
<style>
    body{
        font-family: arial, sans-serif;
    }
</style>
</head>
<body>
<form action="index.php" method="post">
<table cellpadding=20>
<tr>
<td>

<h3>Log-in</h3>
<table border="0" cellpadding="10">
<tr>
<td>User name </td>
<td>
<? echo "<input type=text size=30 name=user >"; ?>
</td>
</tr>
```

```

        <tr>
        <td>Password</td>
        <td>
        <? echo "<input type=password size=30 name=pass >"; ?>
        </td>
        </tr>
        <tr>
        <td></td>
        <td><input type=submit value="Enter"></td>
        </tr>
    </table>
</td>
</tr>
</table>
</form>
</body>
</html>

```

Save the file as **login.php** and copy it to the server. Obtain a suitable illustration, save this with the name **loginImage.jpg** and also copy it to the server.

Run the website, specifying the filename **login.php** at the end of the timeline URL. Check that input boxes are displayed for the user name and password, along with the picture illustration and an 'Enter' button.

We will extend the log-in page to provide a form for new members to register. The overall page structure is constructed from nested tables.



Re-open the **login.php** file and add the program code shown below. This creates input boxes which allow a new user to input their personal details and select a user name and password.

Save the completed file and copy it to the server. Run the website log-in page and check that data entry boxes are displayed correctly.

```

                <td><input type=submit value="Enter"></td>
            </tr>
        </table>
    </td>
</form>
<form method='post' action="addMember.php">
    <td width =200>
        </td>
        <td>
            <table cellpadding=5>
                <tr><td colspan=2>
                    <h3>Register as a new user</h3></td></tr>
                <tr><td>Forename
                    <td><input type=text name='forename'></td></tr>
                <tr><td>Surname</td>
                    <td><input type=text name='surname'></td></tr>
                <tr><td>e-mail</td>
                    <td><input type=text name='email'></td></tr>
                <tr><td><br>Please select: </td></tr>
                <tr><td>User name</td>
                    <td><input type=text name='user'></td></tr>
                <tr><td>Password</td>
                    <td><input type=password name='pass'></td></tr>
                <tr><td></td><td><input type=submit value='Enter'>
                    </td></tr>
            </table>
        </td></tr>
    </table></td>
</tr>
</table>
</form>
</body>
</html>

```

Registration data entered by a new user must be saved to the **timelineLogin** table in the database. To do this we will first create a **TimelineLogin** object class. All transfers of data between the website and the SQL database will be handled by separate object classes. This strategy has the advantages of: simplifying the testing of database functions; reducing the complexity of the program code which remains in the web page files; and producing object classes which can be re-used in different projects.

Open a blank file and add the lines of program code shown below. Save the file as **TimelineLogin.php** and copy it to the server. The class file begins by defining the fields for a TimelineLogin object, corresponding to the fields of the database table. A constructor method is then provided to create objects, and a method **addMember()** allows records to be saved to the database.

```

<?
class TimelineLogin
{
    public static $timelineUser=array();
    public static $userCount;
    private $loginID;
    private $forename;
    private $surname;
    private $email;
    private $user;
    private $pass;
    private $status;

    function __construct($loginID,$forename,$surname,$email,$user,$pass,$status)
    {
        $this->loginID = $loginID;
        $this->forename = $forename;
        $this->surname = $surname;
        $this->email = $email;
        $this->user = $user;
        $this->pass = $pass;
        $this->status = $status;
    }

    public static function addMember($forename,$surname,$email,$user,$pass)
    {
        include('user.inc');
        $conn = new mysqli(localhost, $username, $password, $database);
        if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
        $query="INSERT INTO timelineLogin VALUES ('','$forename',
            '$surname','$email','$user','$pass','member')";
        $result=mysqli_query($conn, $query);
        mysqli_close($conn);
    }
}
?>

```

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.

We will now create the page which will be loaded when a new member enters their personal details on the log-in screen and clicks the 'Enter' button. Open a blank file and add the lines of program code below. The

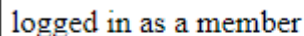
program collects the data values from the input boxes, then calls the **addMember()** method in the class file which will upload the record to the database table.

```
<?
    session_start();
?>
<html>
  <head>
    <title> Historical timeline </title>
  </head>
  <body>
    <?
      $forename=$_REQUEST['forename'];
      $surname=$_REQUEST['surname'];
      $email=$_REQUEST['email'];
      $user=$_REQUEST['user'];
      $pass=$_REQUEST['pass'];
      include('TimelineLogin.php');
      TimelineLogin::addMember($forename,$surname,$email,$user,$pass);
      $_SESSION['login']="member";
      header('Location: index.php');
    ?>
  </body>
</html>
```

Save the file as **addMember.php** and copy it to the server. Open a blank file, add the program code below, save it as **index.php** and copy the file to the server.

```
<?
    session_start();
    $login = $_SESSION['login'];
    if ($login == 'member')
        echo"<p>logged in as a member";
?>
```

We are now ready to test the system for registering new members. Run the website log-in page, enter full details for a new member, then click the 'Enter' button. The page **index.php** should load, and display a message indicating that the user is now logged-in as a member.



Go to the PHP MyAdmin website and open the **timelineLogin** table. The new record should be present, with the status field set to 'member'.

We will now develop the login function for existing staff and members. Open the **TimelineLogin.php** class file and add the **loadTimelineUsers()** method shown below. This accesses the database table and obtains all timeline login records. These are then converted to objects which are identified as `$timelineUser[1]`, `$timelineUser[2]`, etc.

```

public static function loadTimelineUsers()
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM timelineLogin";
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    $i=1;
    while ($i <= $num)
    {
        $row=mysqli_fetch_assoc($result);
        $loginID=$row["loginID"];
        $forename=$row["forename"];
        $surname=$row["surname"];
        $email=$row["email"];
        $user=$row["userName"];
        $pass=$row["password"];
        $status=$row["status"];
        TimelineLogin::$timelineUser[$i] = new TimelineLogin
            ($loginID,$forename,$surname,$email,$user,$pass,$status);
        $i++;
    }
    TimelineLogin::$userCount=$num;
}
}
?>

```

Add two further methods to the **TimelineLogin.php** class file. These check each of the objects for a valid user name and password. If found, the status as 'staff' or 'member' is returned.

```

private function checkUser($userWanted,$passWanted)
{
    if (($userWanted==$this->user)&&($passWanted==$this->pass))
        return $this->status;
    else
        return "";
}

public static function checkPassword($userWanted,$passWanted)
{
    $found="";
    for ($i=1;$i<=TimelineLogin::$userCount;$i++)
    {
        $answer= TimelineLogin::$timelineUser[$i]->checkUser
            ($userWanted,$passWanted);
        if (strlen($answer)>1)
            $found=$answer;
    }
    return $found;
}
}
?>

```

Save the **TimelineLogin.php** file and copy it to the server. Re-open the **index.php** file. Add lines of program code to call the password checking method and obtain the member or staff status of the user. This is then stored as a session variable for later use.

```
<?
  session_start();
  $login = $_SESSION['login'];

  if ($login == '')
  {
    include('TimelineLogin.php');
    $user=$_REQUEST['user'];
    $pass=$_REQUEST['pass'];
    TimelineLogin::loadTimelineUsers();
    $login=TimelineLogin::checkPassword($user,$pass);
    $_SESSION['login']=$login;
    $_SESSION['user']=$user;
  }

  if ($login == 'member')
    echo"<p>logged in as a member";
```

Add an additional test message to identify staff users.

```
  if ($login == 'member')
    echo"<p>logged in as a member";

    else if ($login == 'staff')
      echo"<p>logged in as staff";
    else
      echo"<p>no login";

?>
```

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

Run the website log-in page and carry out tests with **member** and **staff** user names and passwords. In each case, a corresponding log-in message should be displayed when the 'Enter' button is clicked. If an incorrect user name or password is entered, then 'no login' should be shown.

This **index.php** page will display the historical timeline when the program is completed. We will replace the log-in test messages with a menu bar at the top of the screen. In preparation for this, create a style sheet. Open a blank file and add the formatting commands shown below.

```
body, table {
  font-family: arial, sans-serif;
  font-size: 14px;}
table.outline, th.outline, td.outline {
  border-collapse: collapse; border: 1px solid gray;}
table.menu {
  font-family: arial, sans-serif;
  border-collapse: collapse; width: 100%;}
th.menu {
  text-align: left; padding: 8px;
  background-color: rgb(180, 180, 180); color: white;}
a:link, a:visited {
  color: white; font-size: 14px;
  text-decoration: none;}
```


Save the file as **styleSheet.css** and copy it to the server.

Re-open the **index.php** file. Remove the IF..ELSE..ELSE block, then add program code to create menus.

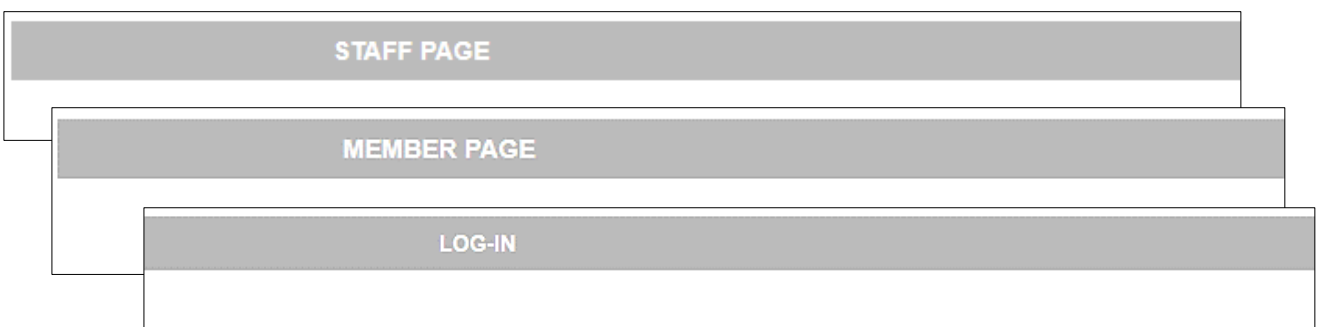
```

        include('TimelineLogin.php');
        $user=$_REQUEST['user'];
        $pass=$_REQUEST['pass'];
        TimelineLogin::loadTimelineUsers();
        $login=TimelineLogin::checkPassword($user,$pass);
        $_SESSION['login']=$login;
        $_SESSION['user']=$user;
    }
?>
<html>
<head>
<title> Historical timeline </title>
<link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
<?
    echo"<table class=menu>";
    echo"<tr><th class=menu></th>";
    echo"<th class=menu>";
    if ($login=='member')
    {
        echo"<a href='memberArticles.php'>";
        echo"MEMBER PAGE </a></th>";
    }
    else if ($login=='staff')
    {
        echo"<a href='createTimeline.php'>";
        echo"STAFF PAGE </a></th>";
    }
    else
    {
        echo"<a href='login.php'>";
        echo"LOG-IN </a></th>";
    }
    echo"</tr>";
?>
</table>
</body>
</html>

```



Save **index.php** and copy it to the server. Run the website and again test the log-in function with **staff**, **member** and incorrect user names and passwords. Menu bars should appear as shown below.




```

?>
<p>&nbsp;&nbsp;&nbsp;&nbsp;<input type=submit value='continue... '>
</form>

<script>
function catChanged()
{
  for (i=1;i<=6;i++ )
  {
    textboxID="cat"+i;
    checkboxID="cat"+i+"check";
    textEntered = document.getElementById(textboxID).value;
    if (textEntered.length<1)
      document.getElementById(checkboxID).checked = false;
    else
      document.getElementById(checkboxID).checked = true;
  }
}
</script>


</body>
</html>

```

Re-save **createTimeline.php** and copy it to the server. Run the website and log-in as staff. Click on 'STAFF PAGE' to reach the 'Set up new timeline' option. Check that the page is displayed correctly.


Before carrying out further work on the program, we must set up tables in the database to store the timeline and timeline category data.

Go to the PHP MyAdmin web page and display the list of existing database tables. Select the 'new' option and add a table with the name **timeline**. Add fields as shown below. The integer **timelineID** field should be specified as the primary key, and set to auto-increment as records are added. The **timelineTitle** field is of data type **varchar**, with a length of 40 characters. The year fields are integers.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	timelineID 	int(11)			No	None		AUTO_INCREMENT
2	timelineTitle	varchar(40)	latin1_swedish_ci		No	None		
3	startYear	int(11)			No	None		
4	finishYear	int(11)			No	None		

Save the **timeline** table.

Create another new table with the name **timelineCategory**. Add fields as shown below. The integer **categoryID** field should be specified as the primary key, and set to auto-increment. The **categoryName** field is of data type **varchar**, with a length of 30 characters. The **timelineID** and **symbol** fields are integers.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	categoryID 	int(11)			No	None		AUTO_INCREMENT
2	timelineID	int(11)			No	None		
3	categoryName	varchar(30)	latin1_swedish_ci		No	None		
4	symbol	int(11)			No	None		

Save the **timelineCategory** table.

When timeline data is entered, it will be stored in the two database tables. We now need to set up **Timeline** and **TimelineCategory** object classes to facilitate the file operations.

Open a blank file and add the lines of program code below to create a **Timeline** class file.

The file begins by defining the attributes for a **Timeline object**, which correspond with the fields of the database table. An array **\$timelines[]** is used to identify the objects as **\$timelines[1]**, **\$timelines[2]**, ...

We then include a **constructor** method, and a method to add a new timeline record to the database. Notice that the **addTimeline()** method returns the **timelineID** value allocated by the auto-number function, which will be needed to link the category values to the correct timeline.

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

```
<?
class Timeline
{
    public static $timelines=array();
    private $timelineID;
    private $timelineTitle;
    private $startYear;
    private $finishYear;

    function __construct($timelineID,$timelineTitle,$startYear,$finishYear)
    {
        $this->timelineID = $timelineID;
        $this->timelineTitle = $timelineTitle;
        $this->startYear = $startYear;
        $this->finishYear = $finishYear;
    }

    public static function addTimeline($timelineTitle,$startYear,$finishYear)
    {
        include('user.inc');
        $conn = new mysqli(localhost, $username, $password, $database);
        if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
        $query="INSERT INTO timeline VALUES ('','$timelineTitle',
        '$startYear','$finishYear')";
        $result=mysqli_query($conn, $query);
        $timelineID = mysqli_insert_id($conn);
        mysqli_close($conn);
        return $timelineID;
    }
}
?>
```

We will now set up the **TimelineCategory** class file. Open a blank file and add the lines of program code below to define the attributes for a TimelineCategory object.

```
<?
class TimelineCategory
{
    public static $categories=array();
    private $categoryID;
    private $timelineID;
    private $categoryName;
    private $symbol;
}
?>
```

Continue by adding a **constructor** method and an **addCategory()** method to the file, as shown below.


```

private $categoryName;
private $symbol;

function __construct($categoryID,$timelineID,$categoryName,$symbol)
{
    $this->categoryID = $categoryID;
    $this->timelineID = $timelineID;
    $this->categoryName = $categoryName;
    $this->symbol = $symbol;
}

public static function addCategory($timelineID,$categoryName,$symbol)
{
    include('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="INSERT INTO timelineCategory VALUES ('','$timelineID',
                                                    '$categoryName','$symbol')";
    $result=mysqli_query($conn, $query);
    mysqli_close($conn);
}
}
?>

```

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

We now have the necessary class methods to save a timeline and event categories. Before entering test data, a page must be created to run these methods when the 'continue' button is clicked on the data entry page. Open a blank file and enter the lines of program code below.

```

<?
    $timelineTitle=$_REQUEST['timeline'];
    $startYear=$_REQUEST['startYear'];
    $finishYear=$_REQUEST['finishYear'];
    $cat[1]=$_REQUEST['cat1'];
    $cat[2]=$_REQUEST['cat2'];
    $cat[3]=$_REQUEST['cat3'];
    $cat[4]=$_REQUEST['cat4'];
    $cat[5]=$_REQUEST['cat5'];
    $cat[6]=$_REQUEST['cat6'];
?>
<html>
<head>
    <title> Historical timeline </title>
</head>
<body>
</body>
</html>

```

The program collects the data values entered on the **createTimeline.php** page and assigns these to PHP variables.

Add further lines of program code to call the **addTimeline()** and **addCategory()** methods which will save the records in the database tables.

```

</head>
<body>
  <?
  include('Timeline.php');
  include('TimelineCategory.php');
  $timelineID = Timeline::addTimeline($timelineTitle,$startYear,$finishYear);
  for ($i=1; $i<=6; $i++)
  {
    if (strlen($cat[$i])>0)
    {
      TimelineCategory::addCategory($timelineID,$cat[$i],$i);
    }
  }
  $address = 'Location: timelineDesign.php?timelineWanted='.$.$timelineID;
  header($address);
  ?>
</body>
</html>

```

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

For test purposes, we will now create a timeline with the title **'History of Transport'**. This will cover the years from **1700** to **2020**, and will include four categories of event: **Road, Sea, Rail** and **Air**.

Run the website and log-in as staff. Select the 'Set up new timeline' option from the staff menu. Enter the test data into the input boxes, then click the 'continue' button.

Do not be concerned that the browser displays a 'URL not found' message. The website is trying to load a page **timelineDesign.php** which we will add shortly.

Go to the PHP MyAdmin web page. Open the **timeline** table and check that a 'History of Transport' record is present. Open the **timelineCategory** table and check that the four records for Road, Sea Rail and Air are present.

Now that the timeline data can be stored successfully in the database, we will add methods to the **Timeline** class file to reload and display the data. Re-open the **Timeline.php** file and add the methods shown below. Save the updated file and copy it to the server.

```

public static function loadTimelines()
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM timeline";
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    $i=1;
    while ($i <= $num)
    {
        $row=mysqli_fetch_assoc($result);
        $timelineID=$row["timelineID"];
        $timelineTitle=$row["timelineTitle"];
        $startYear=$row["startYear"];
        $finishYear=$row["finishYear"];
        Timeline::$timelines[$i] = new Timeline($timelineID,
                                                $timelineTitle,$startYear,$finishYear);
        $i++;
    }
    return $num;
}

public function getTimelineID(){return $this->timelineID;}
public function getTimelineTitle(){return $this->timelineTitle;}
public function getStartYear(){return $this->startYear;}
public function getFinishYear(){return $this->finishYear;}
}
?>

```

It will also be necessary to add methods to the **TimelineCategory** class file to reload records from the database. Open the **TimelineCategory.php** file and add the methods shown in the two boxes below. Save the updated file and copy it to the server.

```

public static function loadByTimelineID($timelineIDwanted)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM timelineCategory WHERE
                                                timelineID = ".$timelineIDwanted;
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    $i=1;
    while ($i <= $num)
    {
        $row=mysqli_fetch_assoc($result);
        $categoryID=$row["categoryID"];
        $timelineID=$row["timelineID"];
    }
}

```

```

$categoryID=$row["categoryID"];
$timelineID=$row["timelineID"];

$categoryName=$row["categoryName"];
$symbol=$row["symbol"];
TimelineCategory::$categories[$i] = new TimelineCategory($categoryID,
                                                         $timelineID,$categoryName,$symbol);
    $i++;
}
return $num;
}

public function getCategoryName(){return $this->categoryName;}
public function getSymbol(){return $this->symbol;}
}
?>

```

We can now return to work on the timeline design page which will allow the historical periods to be created for the timeline. The 'History of Transport' timeline might include periods such as: 'The age of sail' or 'The jet age'.

Open a blank file and add the lines of program code shown in the two boxes below. Save the file as **timelineDesign.php**. The program will load all timeline records from the database table, then display the titles in a drop down list. When a timeline is selected, its title is displayed below as a heading:

View timeline	Set up new timeline	Edit timeline periods
<p>Timeline design</p> <p>Add or edit a historical period on an existing timeline.</p> <p>Select timeline: <input type="text" value="History of Transport"/></p> <p>History of Transport</p>		

```

<html>
<head>
  <title> Historical timeline </title>
  <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
  <?
    include('staffMenu.php');
  ?>
  <h3>Timeline design</h3>
  Add or edit a historical period on an existing timeline.
  <br><br>
  Select timeline:
  <?
    $timelineWanted=$_REQUEST['timelineWanted'];
    include ('Timeline.php');
    $num = Timeline::loadTimelines();

```

```

for ($k=1; $k<=$num;$k++)
{
    $AtimelineID[$k]=Timeline::$timelines[$k]->getTimelineID();
    $AtimelineTitle[$k]=Timeline::$timelines[$k]->getTimelineTitle();
    $AstartYear[$k]=Timeline::$timelines[$k]->getStartYear();
    $AfinishYear[$k]=Timeline::$timelines[$k]->getFinishYear();
}
echo"<select name=timelineWanted ID=timelineWanted onChange=changeTimeline()>";
echo"<option>";
for ($k=1; $k<=$num;$k++)
{
    if ($timelineWanted==$AtimelineID[$k])
    {
        echo"<option selected value=".$AtimelineID[$k].">".$AtimelineTitle[$k];
        $timelineTitleWanted = $AtimelineTitle[$k];
        $startYearWanted=$AstartYear[$k];
        $finishYearWanted=$AfinishYear[$k];
    }
    else
    echo"<option value=".$AtimelineID[$k].">".$AtimelineTitle[$k];
}
echo"</select>";
echo "<h3>".$timelineTitleWanted."</h3>";
?>
<script>
function changeTimeline()
{
    timelineIDwanted = document.getElementById("timelineWanted").value;
    window.location = "timelineDesign.php?timelineWanted="+timelineIDwanted;
}
</script>
</body>
</html>

```

The next sections of the page will display the start and finish years for the selected timeline, and list the timeline categories. Go to the **timelineDesign.php** file and add the program code below.

```

    echo"<option value=".$AtimelineID[$k].">".$AtimelineTitle[$k];
}
echo"</select>";
echo "<h3>".$timelineTitleWanted."</h3>";

```

```

include ('TimelineCategory.php');
$catCount=TimelineCategory::loadByTimelineID($timelineWanted);
for ($i=1;$i<=$catCount;$i++)
{
    $catName[$i]=TimelineCategory::$categories[$i]->getCategoryName();
}
echo"<table class=outline cellpadding=10>";
echo"<tr><td>Start year:</td>";
echo"<td><input type=text size=10 name=startYear value=$startYearWanted></td>";
echo"<td>&nbsp;</td></tr>";
echo"<tr><td>Finish year:</td>";
echo"<td><input type=text size=10 name=finishYear value=$finishYearWanted></td>";
echo"<td>&nbsp;</td></tr>";
echo"</table>";
echo"<br><br>";
echo"<table class=outline cellpadding=5>";
echo"<tr><td>Event categories</td>";
echo"<td>&nbsp;</td></tr>";

```

```

echo"<tr><td>Event categories</td>";
echo"<td>&nbsp;&nbsp;&nbsp;</td></tr>";

for ($m=1;$m<=6; $m++)
{
    echo"<tr><td class=plain><input type=text size=30
        name='cat'."$m."'' value='$catName[$m]''></td>";
    echo"<td class=plain>&nbsp;&nbsp;&nbsp;</td></tr>";
}
echo"</table>";
echo"<br><br>";
echo"<form method=post action='addPeriod.php?timeline=".$timelineWanted."'>";
echo"<input type=submit value='add another historical period'>";
echo"</form>";

?>
<script>
function changeTimeline()
{
    timelineIDwanted = document.getElementById("timelineWanted").value;
    window.location = "timelineDesign.php?timelineWanted="+timelineIDwanted;
}
</script>

```

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

Run the website and log-in as staff. Select the 'Edit timeline periods' option from the staff menu. Use the drop-down list box to select the History of Transport timeline. Check that the year range and timeline categories are displayed correctly, as shown below.

Timeline design

Add or edit a historical period on an existing timeline.

Select timeline:

History of Transport

Start year:

Finish year:

Event categories

The next step in developing the website is to set up the historical periods which will make up the timeline. Before doing this, it will be necessary to create a database table to store the data. Open the PHP MyAdmin web page, list the existing tables and select the 'New' option. Set up a table with the name **timelinePeriod** and add fields as shown below.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	periodID	int(11)			No	None		AUTO_INCREMENT
2	timelineID	int(11)			No	None		
3	startYear	int(11)			No	None		
4	finishYear	int(11)			No	None		
5	periodName	varchar(30)	latin1_swedish_ci		No	None		
6	imageName	varchar(30)	latin1_swedish_ci		No	None		
7	yearPixels	int(11)			No	20		

The integer **periodID** field should be specified as the primary key, and set to auto-increment. The **periodName** and **imageName** fields are of data type **varchar** with a length of 30 characters. The **startYear**, **finishYear** and **yearPixels** fields are integers. Set a default value of '20' for **yearPixels**.

We will now create a class file to facilitate transfer of timeline period records between the database and the web page. Open a blank file and add the lines of program code below. These define the attributes for a **TimelinePeriod** class, corresponding to the fields of the table. A **constructor** method is provided, and an **addPeriod()** method to save a timeline period record into the database.

```

<?
class TimelinePeriod
{
    public static $periods=array();
    private $periodID;
    private $timelineID;
    private $startYear;
    private $finishYear;
    private $periodName;
    private $imageName;
    private $yearPixels;

    function __construct($periodID,$timelineID,$startYear,$finishYear,
                        $periodName,$imageName,$yearPixels)
    {
        $this->periodID = $periodID;
        $this->timelineID = $timelineID;
        $this->startYear = $startYear;
        $this->finishYear = $finishYear;
        $this->periodName = $periodName;
        $this->imageName = $imageName;
        $this->yearPixels = $yearPixels;
    }

    public static function addPeriod($timelineID,$startYear,$finishYear,
                                    $periodName,$imageName,$yearPixels)
    {
        include('user.inc');
        $conn = new mysqli(localhost, $username, $password, $database);
        if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
        $query="INSERT INTO timelinePeriod VALUES ('','$timelineID',
                                                    '$startYear', '$finishYear','$periodName',
                                                    '$imageName','$yearPixels')";

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

```

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

Historical periods can now be added to the timeline. This will involve a series of processes, as illustrated in the flowchart shown here.

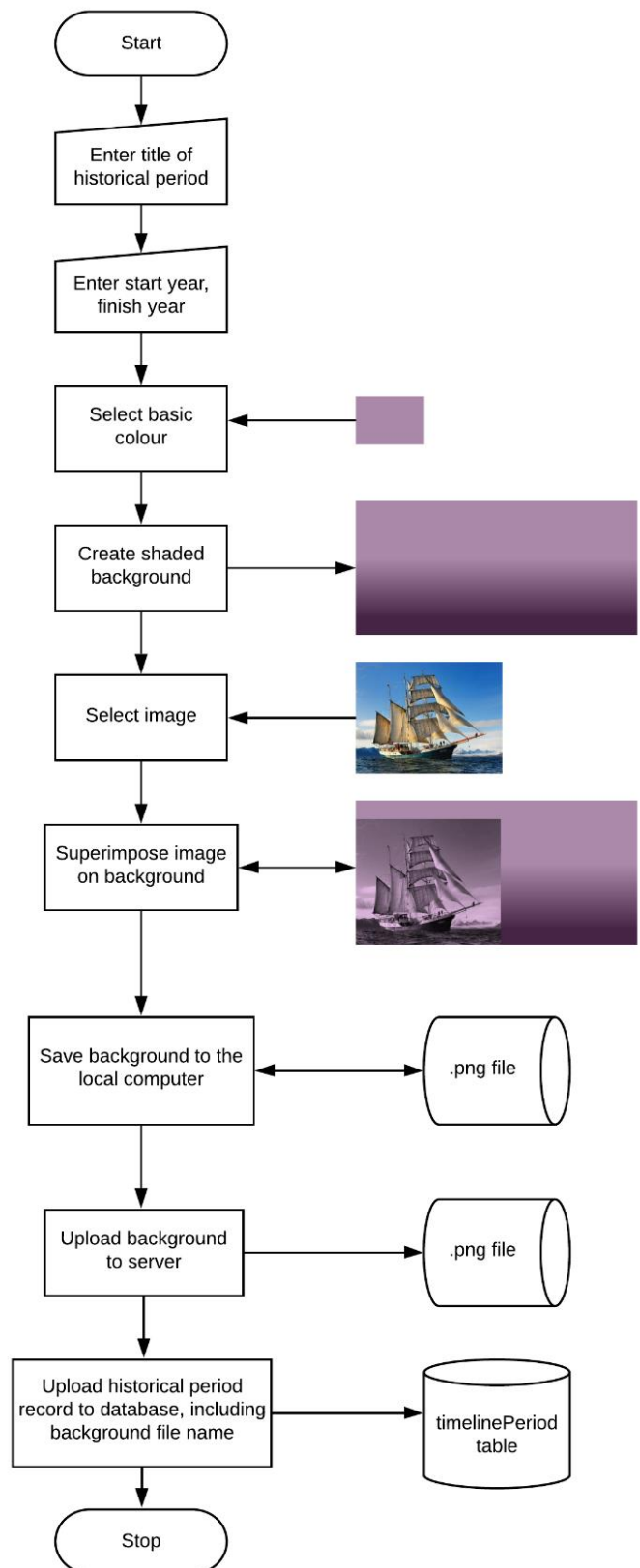
The title of the historical period, its start and finish years are entered, e.g. The Age of Sail, 1700 – 1840.

The next step will be to create a background image for the historical period when it is displayed on the timeline web page. We begin by selecting a basic colour from a colour chart. This is then used to create a shaded rectangle.

A photographic image is selected which is representative of the historical period. The computer will then superimpose the picture on the background, using a colour palate linked to the background colour.

The completed background is saved firstly to the local computer in **.png** graphics format, then uploaded to the server.

The final step is to save a text record containing the period title, year range and file name for the background image. The record is uploaded to the **timelinePeriod** table in the database.



We will begin by creating a web page to input the period title and year range, and to control the production of the background image. Open a blank file and add the lines of program code shown below.

```

<?
    $timelineWanted = $_REQUEST["timeline"];
?>
<html>
<head>
    <title> Historical timeline </title>
    <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
<?
    include('staffMenu.php');
    include ('Timeline.php');
    $count= Timeline::loadTimelines();
    $timelineTitleWanted=Timeline::loadTitleByID($timelineWanted,$count);
    echo"<br><form method=post action='uploadBackground.php?timelineID=
        $timelineWanted' enctype='multipart/form-data'>";
    echo"<h3>".$timelineTitleWanted."</h3>";
    echo"Add a historical period:";
    echo"<br><br>Period title";
    echo"&nbsp;&nbsp;&nbsp;&nbsp;<input type=text size=30 name=periodTitle
        id=periodTitle value='$periodTitle'>";
    echo"<br><br>Begins in year";
    echo"&nbsp;&nbsp;&nbsp;&nbsp;<input type=text size=10 name=startYear
        id=startYear value=$startYear>";
    echo"<br><br>Ends in year";
    echo"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<input type=text size=10 name=finishYear
        id=finishYear value=$finishYear>";
    echo"<br><br><br>";
    echo"<table class=outline cellpadding=10>";
    echo"<tr><td><b>Step 1:</b> Click to design a background image which
        you will store on your computer";
    echo"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<input type=button value=
        'Design background' onClick='newBackground()'></td></tr>";
    echo"<tr><td><b>Step 2:</b> The background image can now be uploaded
        to the server.<br>";
    echo"Select the background which you designed and stored: <input type=
        'file' name='fileToUpload' id='fileToUpload'></td></tr>";
    echo"</table>";
    echo"<br><br><input type=submit value ='Finished' >";
    echo"<br><br>";
    echo"</form>";
?>
</body>
</html>

```

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

The objective of the program code is to display an input form and instructions for the user, as shown below.

View timeline	Set up new timeline	Edit timeline periods	Approve or edit events
<h3>History of Transport</h3> <p>Add a historical period:</p> <p>Period title <input type="text"/></p> <p>Begins in year <input type="text"/></p> <p>Ends in year <input type="text"/></p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Step 1: Click to design a background image which you will store on your computer <input type="button" value="Design background"/></p> <p>Step 2: The background image can now be uploaded to the server. Select the background which you designed and stored: <input type="button" value="Choose file"/> No file chosen</p> </div> <p><input type="button" value="Finished"/></p>			

Before running the page, it will be necessary to add a small method to the Timeline class file which will input the timeline ID number and return the corresponding timeline title.

Open **Timeline.php** and add the method shown below. Save the file and copy it to the server.

```

public function getStartYear(){return $this->startYear;}
public function getFinishYear(){return $this->finishYear;}

public static function loadTitleByID($timelineIDwanted,$count)
{
    $title="";
    for ($i=1; $i<=$count; $i++)
    {
        if (Timeline::$timelines[$i]->getTimelineID() == $timelineIDwanted)
            $title = Timeline::$timelines[$i]->getTimelineTitle();
    }
    return $title;
}
}
?>
    
```

Run the website and log-in as staff. Go to 'staff page' and select the 'Edit timeline periods' option from the staff menu. Choose the 'History of Transport' timeline from the drop-down list, then click the 'add another historical period' button. Check that the page display is displayed correctly as shown above, including the title 'History of Transport'.

View timeline	Set up new timeline	Edit timeline periods
<div style="border: 1px solid black; border-radius: 15px; padding: 5px; display: inline-block; margin-bottom: 5px;"> History of Transport </div> <p>Add a historical period:</p>		

We can now move on to the next stage in setting up the timeline. After entering a title and year range for the historical period, the user will click the 'Design background' button. To activate this, re-open the **addPeriod.php** file and add a JavaScript function as shown. This function will collect values from the title and year range input boxes, and attach them to the URL for a page which will create the background image.

```

    echo"<br><br><input type=submit value ='Finished' >";
    echo"<br><br>";
    echo"</form>";
?>
<script>
    function newBackground()
    {
        periodTitle = document.getElementById("periodTitle").value;
        startYear = document.getElementById("startYear").value;
        finishYear = document.getElementById("finishYear").value;
        address = "newBackground.php?periodTitle="+periodTitle+
            "&startYear="+startYear+"&finishYear="+finishYear;
        window.location = address;
    }
</script>
</body>
</html>

```

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

The next page will allow the user to select a basic colour from the standard HTML colour picker component, then use this to create the shaded background rectangle.



For this and other graphics tasks during the historical timeline project, the language **p5.js** will be used. This is not a programming language in its own right, but rather a high level extension of JavaScript. Processes can be written very simply in **p5.js** code, for example: drawing a circle with a particular size, colour and screen position; or finding the X and Y co-ordinates of the mouse and whether a mouse button is being pressed. The **p5.js** code will then be translated into a sequence of JavaScript commands which actually carry out the task. Any process written in **p5.js** code could be written directly in native JavaScript and would operate in exactly the same way on screen. However, it would probably require many more lines of programming code, would be more complex to understand, and there would be a greater chance of the programmer making an error.

In order to use the **p5.js** language, two files must be uploaded to the server and placed in the **timeline** folder. The files are:

p5.js which handles graphics and other programming commands
p5.dom.js which allows HTML components to be included with graphics

Both of these files are available for free download from the developers' web site: p5js.org

Obtain copies of **p5.js** and **p5.dom.js** and upload them to the server. We can then continue with the program development. Open a blank file and add the lines of program code below.

```

<?
  session_start();
  $timelineWanted=$_SESSION["timeline"];
  $periodTitle = $_REQUEST["periodTitle"];
  $startYear = $_REQUEST["startYear"];
  $finishYear = $_REQUEST["finishYear"];
  $_SESSION["periodTitle"]=$periodTitle;
  $_SESSION["startYear"]=$startYear;
  $_SESSION["finishYear"]=$finishYear;
?>
<html>
<head>
  <title> Historical timeline </title>
  <script src="p5.js"></script>
  <script src="p5.dom.js"></script>
  <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
  <?
    include('staffMenu.php');
  ?>
  <h3>Background design</h3>
  Click on the coloured button below to adjust the background shading,
  then click to continue.
  <form method=post action='selectImage.php?imageFile=blank.jpg'>
  <br><input type=submit value='continue...'>
  </form>
</body>
</html>

```

The program begins by collecting the timeline ID, period title and year range, then saves them as **session variables** for future use. The p5.js translation files are then loaded.

The next step is to create a graphics canvas, and add a colour picker component. As with JavaScript, the p5.js code is included within a **<script>** block. Insert the lines of program code below.

```

<form method=post action='selectImage.php?imageFile=blank.jpg'>
<br><input type=submit value='continue...'>
</form>

<script>
  var inp1;
  function setup()
  {
    createCanvas(1200,540);
    pixelDensity(1);
    inp1 = createColorPicker('#aa88aa');
    inp1.position(20, 160);
  }
</script>

</body>
</html>

```

Save the file as **newBackground.php** and copy it to the server. Run the website as previously, logging-in as staff and go to the Staff Page.

Select the 'Edit timeline periods' option and choose the History of Transport timeline from the drop-down list. Click the button to add another historical period. Enter details for a historical period with the title '**Age of Sail**' and a year range of **1700** to **1840**.

Click the 'Design background' button. When the new page is loaded, select the colour picker button. A colour choice window should open.

Period title	<input type="text" value="Age of Sail"/>
Begins in year	<input type="text" value="1700"/>
Ends in year	<input type="text" value="1840"/>



Return to the **newBackground.php** file. We will now add p5.js code to create a shaded rectangle based on the colour selected in the colour picker window. Add the **draw()** function shown below.

```
function setup()
{
  createCanvas(1200,540);
  pixelDensity(1);
  inp1 = createColorPicker('#aa88aa');
  inp1.position(20, 160);
}
```

```
function draw()
{
  background(255);
  noStroke();
  fill(inp1.color());
  rect(20, 40, 960, 460);
  loadPixels();
  var index = (180 + (200 * width)) * 4;
  var r = pixels[index]+1;
  var g = pixels[index+1]+1;
  var b = pixels[index+2]+1;
  localStorage.setItem('red', r);
  localStorage.setItem('green', g);
  localStorage.setItem('blue', b);
  for (i=0;i<=100 ;i++ )
  {
    r=r-1;
    g=g-1;
    b=b-1;
    if (r<0)
      r=0;
    if (g<0)
      g=0;
    if (b<0)
      b=0;
    fill(r,g,b);
    rect(20, 240+2*i, 960, 60);
  }
}
```

```
</script>
```

Save the **newBackground.php** file and copy it to the server. Refresh the page in the browser. A shaded rectangle should now appear. Its colour can be changed by means of the colour picker component. For the historical timeline, it will be best to choose dark backgrounds for the historical periods. This can be done easily by moving just the lower slider on the colour chart.



The **draw()** function begins by obtaining the red, green and blue values (0-255) for the colour selected in the colour picker window. A loop then operates, moving downwards and drawing a series of coloured stripes across the screen. After each stripe is drawn, the red, green and blue values are reduced by 1. This causes the next stripe to be very slightly darker in colour.

Following the sequence outlined in the flowchart on page 208, the next step is to add a picture image to the background. This will be done on another web page. Before continuing with the programming, obtain a photograph of a sailing ship and save this on your computer in **.jpg** or **.png** format.

Open a blank file and add the lines of program code shown in the two boxes below. Save the file as **selectImage.php** and copy it to the server.

```
<?
    session_start();
    $imageFile=$_REQUEST['imageFile'];
    $upload=$_REQUEST['upload'];
?>
<html>
<head>
    <title> Historical timeline </title>
    <script src="p5.js"></script>
    <script src="p5.dom.js"></script>
    <link rel="Stylesheet" type="text/css" href="styleSheet.css" />
</head>
```


Another page will be required to upload of the photograph. Open a blank file and add the program code below. Save the file as **upload.php** and copy it to the server.

```
<?
$imageFile = $_REQUEST['imageFile'];
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file))
{
    echo "The file " . basename( $_FILES["fileToUpload"]["name"]). "
                                     has been uploaded.";
}
$imageFile = basename( $_FILES["fileToUpload"]["name"]);
header("Location: selectImage.php?imageFile=$imageFile&upload=YES");
?>
```

Create a sub-folder within the **timeline** folder on the server. Give this the name **uploads**.

Return to the web browser. Click the '**Choose file**' button and select the sailing ship picture, then click the '**Upload image**' button. Refresh the list of files in the server FTP program, then check that the sailing ship picture file is present in the **uploads** sub-folder within the **timeline** folder.

Return now to the **selectImage.php** file. Add the p5.js program code shown in the two boxes below. Save the file and copy it to the server.

```
<tr><td colspan=3><b>Step 3:</b> Click the 'finished' button to return
    to the Add Historical Period page and upload your saved background file.
    <input type=submit value='finished'>
</form>
</td></tr>
</table>
```

```
<script>
function preload()
{
    var imageWanted="uploads/<? echo $imageFile ?>";
    img1=loadImage(imageWanted);
    img1.filter(GRAY);
    saved=false;
}
function setup()
{
    createCanvas(1200,450);
    pixelDensity(1);
}
function draw()
{
    var r=localStorage.getItem('red');
    var g=localStorage.getItem('green');
    var b=localStorage.getItem('blue');
    background(155);
    noStroke();
    fill(r,g,b);
    rect(0, 0, 1200, 450);
    for (i=0;i<=100 ;i++ )
    {
        r=r-1;
        g=g-1;
```



```

        r=r-1;
        g=g-1;
    }
    b=b-1;
    if (r<0)
        r=0;
    if (g<0)
        g=0;
    if (b<0)
        b=0;
    fill(r,g,b);
    rect(0, 200+2*i, 1200, 60);
}
}
</script>
</body>
</html>

```

Refresh the web page in the browser. The shaded background rectangle should be displayed as previously. We will now add program code to display the picture image.

Return to the **selectImage.php** file and add lines of code to the **draw()** function as shown below. Also add an **uploadImage()** function. Save the **selectImage.php** file and copy it to the server.

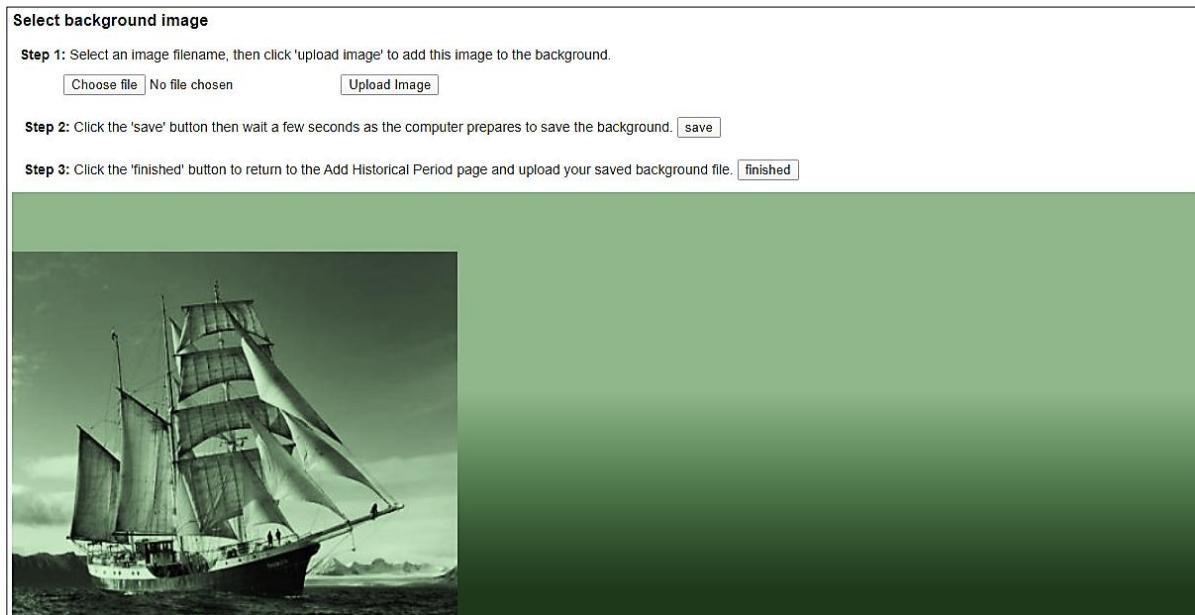
```

        if (b<0)
            b=0;
        fill(r,g,b);
        rect(0, 200+2*i, 1200, 60);
    }
    image(img1,0,60,450,400);
    img1.filter(GRAY);
    loadPixels();
    for (y=60;y<=450 ;y++ )
    {
        for (x=0;x<450 ;x++ )
        {
            index = ((y*width)+x) * 4;
            rpix = pixels[index + 0]+1;
            gpix = pixels[index + 1]+1;
            bpix = pixels[index + 2]+1;
            rp= int(rpix * r /80);
            gp= int(gpix * g /80);
            bp= int(bpix * b /80);
            fill(rp,gp,bp,100);
            rect(x,y,1,1);
        }
    }
}

function uploadImage()
{
    if (saved==false)
    {
        saved=true;
        saveCanvas();
    }
}
}
</script>
</body>
</html>

```

Refresh the browser window. Wait for a few seconds, then the background should re-appear with the sailing ship picture superimposed, as shown below.



The program uploaded the photograph, then converted it to a grey scale image using the **filter(GRAY)** command. A loop then obtained the grey value of each photograph pixel and scaled it according to the red, green and blue components of the background colour. Where a pixel was light grey in the photograph, a light colour was generated, whilst a dark grey pixel generated a darker version of the base colour. Finally, the coloured pixel was copied onto the background to create the superimposed image.

In the browser window, click the **'save'** button shown for Step 2 of the instructions. Be patient as the computer does a lot of work to create a .png file from the finished background image. A file save dialog window will then appear. Select a folder on your computer and save the file as **'age of sail.png'**.

Click the **'finished'** button to return to the 'Add a historical period' page. Currently the input boxes are blank. It is necessary to reload session variables and display these in the boxes. Re-open the **addPeriod.php** file and add the lines of code at the start of the program as shown below.

```

<?
    session_start();
    $startYear = $_SESSION["startYear"];
    $finishYear = $_SESSION["finishYear"];
    $periodTitle = $_SESSION["periodTitle"];

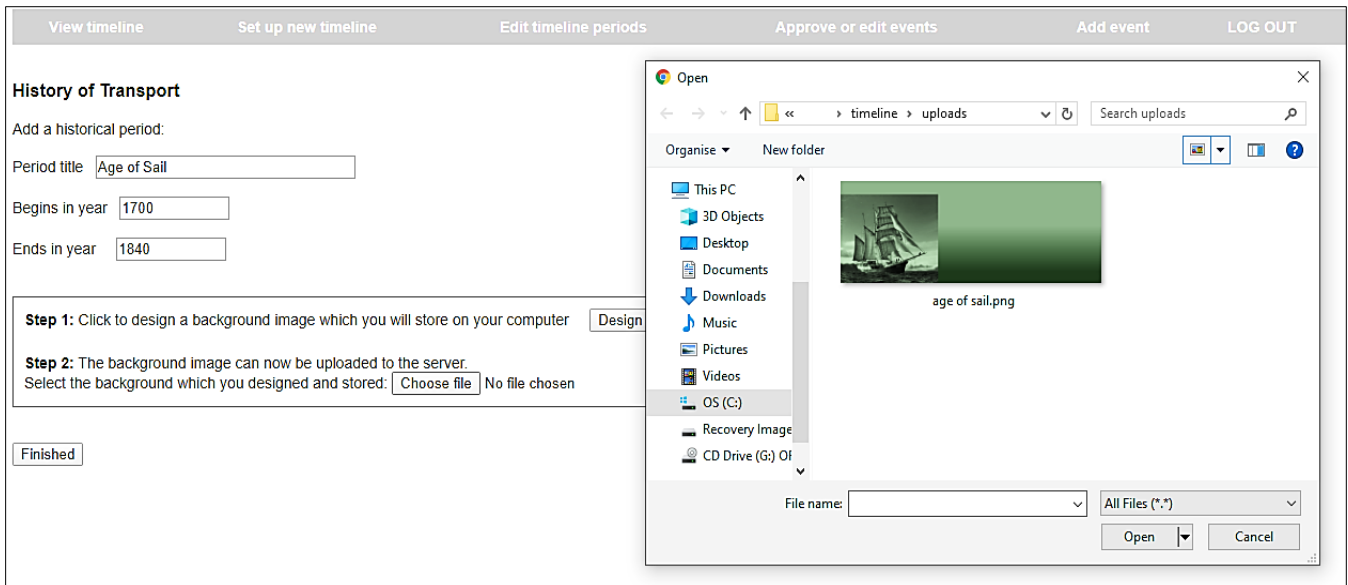
    $timelineWanted = $_REQUEST["timeline"];

    if ($timelineWanted>0)
        $_SESSION["timeline"]=$timelineWanted;
    else
        $timelineWanted=$_SESSION["timeline"];
?>
<html>
<head>
    <title> Historical timeline </title>

```

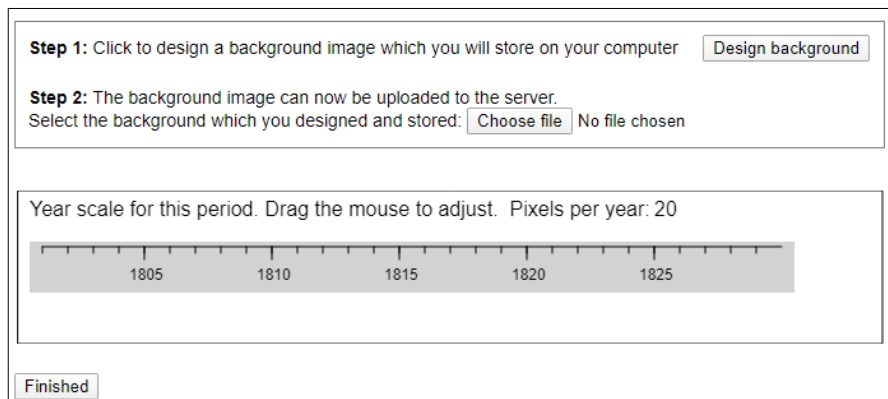
Save **addPeriod.php** and copy it to the server. Refresh the web page.

The historical period title and year range entered earlier should now be displayed. Click the '**Choose file**' button at Step 2 of the instructions, and check that the combined background image has been saved on your computer.



We have another component to add to the page. This is an interactive scale for setting the number of pixels representing each year of the historical period. It may be the case that a period represents a considerable number of years with few historical events to display, such as the Dark Ages. Conversely, many events may need to be recorded for a short span of years, such as the Second World War. Adjusting the year scales for different periods will even out the display of events along the timescale.

The year scale will begin by displaying a default value of 20 screen pixels per year. It will then be possible to adjust the year spacing by dragging the mouse along the scale.



Programming for the year scale will require a range of graphics commands, so p5.js will be used.

Re-open the **addPeriod.php** file. Go to the <head> section and add links to the **p5.js** files.

```
<html>
  <head>
    <title> Historical timeline </title>
    <link rel="stylesheet" type="text/css" href="styleSheet.css" />
    <script src="p5.js"></script>
    <script src="p5.dom.js"></script>
  </head>
```

It is necessary to create a blank space on the page where the graphics can be displayed. Add a loop to insert `
` characters between Step 2 of the instructions and the 'Finished' button.

```

echo"<tr><td><b>Step 2:</b> The background image can now be uploaded
                                to the server.<br>";
echo"Select the background which you designed and stored: <input type=
                                'file' name='fileToUpload' id='fileToUpload' >";
echo"</table>";
for ($s=1; $s<=9; $s++)
{
    echo"<br>";    //timeline scale goes here
}
echo"<br><br><input type=submit value ='Finished' >";
echo"<br><br>";
echo"</form>";
    
```

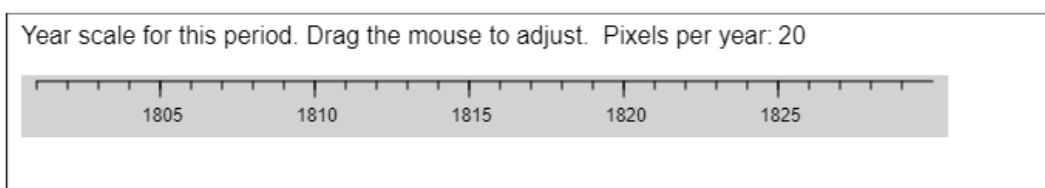
Go now to the `<script>` block at the bottom of the program listing. After the `newBackground()` JavaScript function, add a `yearPixels` variable and the `setup()` function, as shown below.

The `setup()` function will be run by `p5.js` when the page loads. It creates a drawing canvas with the specified width and height, then inserts it into the web page at the required position.

```

<script>
function newBackground()
{
    periodTitle = document.getElementById("periodTitle").value;
    startYear = document.getElementById("startYear").value;
    finishYear = document.getElementById("finishYear").value;
    address = "newBackground.php?periodTitle="+periodTitle+"&startYear="
                                +startYear+"&finishYear="+finishYear;
    window.location = address;
}
var yearPixels = 20;
function setup()
{
    var y=400;
    let cnv = createCanvas(680, 120);
    cnv.position(10, y);
    dragging=false;
}
</script>
</body>
</html>
    
```

We will now produce the year scale. This will have a default value of 20 pixels per year.



The program uses a loop to draw the year divisions on the time scale, using the current value of `yearPixels` to determine the spacing. A MOD operator, represented in p5.js by the `%` symbol, identifies year locations exactly divisible by 5 and adds captions at this points.

The **p5.js** programming system has been designed to simplify graphics animation on web pages. Central to this concept is the **draw()** function which refreshes the graphics display many times a second. Each time **draw()** is automatically called, values of variables such as the X and Y position of the mouse pointer are checked.

Add a **draw()** function below the **setup()** function as shown below. Save the file and copy it to the server. Refresh the web page.

```
function setup()
{
  var y=400;
  let cnv = createCanvas(680, 120);
  cnv.position(10, y);
  dragging=false;
}

function draw()
{
  background(255);
  fill(255);
  stroke(0);
  rect(1,0,678,120);
  textSize(16);
  fill(0);
  noStroke();
  text('Year scale for this period. Drag the mouse to adjust.
                                             Pixels per year:',10,20);

  fill(210);
  rect(10,40,600,40);
  x=20;
  y=44;
  stroke(0);
  line(x,y,x+580,y);
  var i=0;
  var yearStart=1800;
  for (xpos=x;xpos<=(x+560) ;xpos = xpos+yearPixels )
  {
    i++;
    stroke(0);
    line(xpos,y,xpos,y+5);
    if(i%5== 0)
    {
      stroke(0);
      line(xpos,y,xpos,y+10);
      yearDate=yearStart+i;
      fill(0);
      noStroke();
      textSize(12);
      text(yearDate,xpos-12,y+26);
    }
  }
  textSize(16);
  noStroke();
  text(int(yearPixels),500,20);
}

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

In the next programming step, we will interactively adjust the **yearPixels** value by dragging the mouse. This in turn will cause the year scale to be plotted with a different interval when the **draw()** function is called again.

Return to the **addPeriod.php** file. We will now add the interactive mouse function to the **draw()** procedure. Add the lines of program code shown below.

```

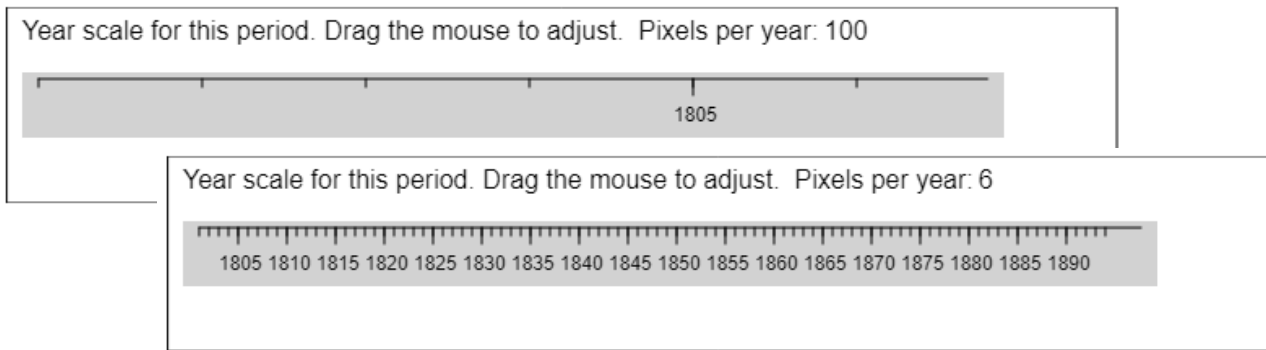
        if(i%5== 0)
        {
            stroke(0);
            line(xpos,y,xpos,y+10);
            yearDate=yearStart+i;
            fill(0);
            noStroke();
            textSize(12);
            text(yearDate,xpos-12,y+26);
        }
    }

    if (mouseIsPressed==true)
    {
        if ((mouseY>10)&&(mouseY<100))
        {
            if (dragging==false)
            {
                dragging=true;
                oldX=mouseX;
            }
            change=mouseX-oldX;
            yearPixels=yearPixels+change/4;
            if (yearPixels<6)
                yearPixels=6;
            if (yearPixels>100)
                yearPixels=100;
            oldX=mouseX;
        }
    }
    else
    {
        dragging=false;
    }

    textSize(16);
    noStroke();
    text(int(yearPixels),500,20);
}
</script>
</body>
</html>

```

Save the **addPeriod.php** file and copy it to the server. Refresh the web page. Drag the mouse pointer sideways on the grey shaded area of the timescale. It should be possible to adjust the year spacing over the range from 6 to 100 pixels per year, as shown below.



This completes the input of all the required data for a historical period, and the record can now be saved into the database. On clicking the '**Finished**' button, a web page will be loaded which will carry out several functions:

- The period **title**, **start year** and **finish year** will be obtained from the text input boxes by means of `$_REQUEST[]` commands.
- The **timelineID** value stored as a session variable will be obtained using a `$_SESSION[]` command.
- An image name will be created by adding the timelineID to the file name of the background image, for example: victorianPeriod_3.png. This will avoid any ambiguity if the same background image name is used in more than one timeline. The background image will then be uploaded to a **backgrounds** folder on the server.
- The **yearPixel** value will be retrieved.
- Finally, the **addPeriod()** method in the **TimelinePeriod** class file will be called to save the record into the database.

In preparation for saving the image file, go to the server and create a **backgrounds** sub-folder within the timeline folder.

A convenient way to store the **yearPixels** value is in a hidden input box. Go to the **addPeriod.php** file and insert this component before the end of the form.

```

for ($s=1; $s<=9; $s++)
{
    echo"<br>"; //timeline scale goes here
}
echo"<br><br><input type=submit value ='Finished' >";
echo"<br><br>";
echo"<input type='hidden' name='yearPixels' id='yearPixels' >";
echo"</form>";
?>
<script>
    function newBackground()

```

It is then just necessary to set the value of **yearPixels** when the time scale is adjusted. Add a line of code at the end of the **draw()** function.

```

    textSize(16);
    noStroke();
    text(int(yearPixels),500,20);
    document.getElementById("yearPixels").value = yearPixels;
}
</script>
</body>

```

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

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

```
<?
    session_start();
    $timeline = $_SESSION["timeline"];
    $startYear = $_REQUEST["startYear"];
    $finishYear=$_REQUEST['finishYear'];
    $periodTitle=$_REQUEST['periodTitle'];
    $yearPixels=$_REQUEST['yearPixels'];
    $yearPixels=intval($yearPixels);
    $target_dir = "backgrounds/";
    $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
    $target_file =substr($target_file,0,-4);
    $target_file = $target_file.'_'.$timeline.'.png';
    $imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
    move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file);
    $imageName = substr($target_file,12);
    include ('TimelinePeriod.php');
    TimelinePeriod::addPeriod($timeline,$startYear,$finishYear,
                                $periodTitle,$imageName,$yearPixels);
    $addressString="Location: timelineDesign.php?timelineWanted=".$timeline;
    header($addressString);
?>
```

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

We can now test the complete procedure for adding a timeline period. Run the website and log-in as staff. Select the 'Edit timeline periods' option from the menu. Choose the History of Transport timeline, then click 'add another historical period'. Enter test data:

- period title 'Age of Sail'
- year range 1700 – 1840.

Select the sailing ship background prepared earlier, then adjust the year scale to 12 pixels per year. When data entry is complete, click the 'Finished' button. The program should return to the Timeline Design page.

Go to the PHP MyAdmin web page and open the timelinePeriod table. Check that the record has been uploaded correctly. Also check on the server that the background image is in the backgrounds folder.

We will now add a table to the Timeline Design page to display details of the timeline periods entered and their background images. Open the **timelineDesign.php** file and add the lines of program code shown below.

```
for ($m=1;$m<=6; $m++)
{
    echo"<tr><td class=plain><input type=text size=30 name='cat'."$m.
                                "' value='$catName[$m]'">";
    echo"<td class=plain>&nbsp;";
}
echo"</table>";
echo"<br><br>";

echo"<table>";
echo"<tr><th class=outline width=300>Period </th>";
echo"<th class=outline width=120>Begin </th>";
echo"<th class=outline width=120>End </th>";
echo"<th class=outline>Background image </th>";
echo"</tr></table>";
echo"<br><br>";

echo"<form method=post action='addPeriod.php?timeline=".$timelineWanted."'>";
echo"<input type=submit value='add another historical period'">";
```


Save the **timelineDesign.php** file and copy it to the server, then refresh the web page. Headings should now be displayed at the bottom of the page, ready to create the table of historical periods.

Event categories

Period	Begin	End	Background image

To obtain data for the table, we must add methods to the **TimelinePeriod** class file to load and access the period records for a particular timeline, as identified by its timelineID value.

Open the **TimelinePeriod.php** file. Add the **loadByTimelineID()** method and the set of **get()** methods which will allow the object attributes to be displayed on the web page.

```

public static function loadByTimelineID($timelineIDwanted)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM timelinePeriod WHERE timelineID="
        $timelineIDwanted." ORDER BY startYear";
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    $i=1;
    while ($i <= $num)
    {
        $row=mysqli_fetch_assoc($result);
        $periodID=$row["periodID"];
        $timelineID=$row["timelineID"];
        $startYear=$row["startYear"];
        $finishYear=$row["finishYear"];
        $periodName=$row["periodName"];
        $imageName=$row["imageName"];
        $yearPixels=$row["yearPixels"];
        TimelinePeriod::$periods[$i]=new TimelinePeriod($periodID,$timelineID,
            $startYear,$finishYear,$periodName,$imageName,$yearPixels);
        $i++;
    }
    return $num;
}
public function getPeriodName(){return $this->periodName;}
public function getPeriodID(){return $this->periodID;}
public function getStartYear(){return $this->startYear;}
public function getFinishYear(){return $this->finishYear;}
public function getImageName(){return $this->imageName;}
public function getYearPixels(){return $this->yearPixels;}
}
?>

```

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

Return to the **timelineDesign.php** file and add a loop to create the table display as shown below. Save the file and copy it to the server. Refresh the Timeline Design web page.


```

echo"<th class=outline width=120>Begin </th>";
echo"<th class=outline width=120>End </th>";
echo"<th class=outline>Background image </th>";

include ('TimelinePeriod.php');
$count = TimelinePeriod::loadByTimelineID($timelineWanted);
for ($i=1; $i<=$count; $i++)
{
    echo"<tr><td class=outline>".TimelinePeriod::$periods[$i]->getPeriodName()."</td>";
    echo"<td class=outline>".TimelinePeriod::$periods[$i]->getStartYear()."</td>";
    echo"<td class=outline>".TimelinePeriod::$periods[$i]->getFinishYear()."</td>";
    echo"<td class=outline ><img src='backgrounds/"
        TimelinePeriod::$periods[$i]->getImageName()." ' width=300></th>";
}

echo"</tr></table>";
echo"<br><br>";
echo"<form method=post action='addPeriod.php?timeline=".$timelineWanted.">";
    
```

An entry should now appear in the table for the Age of Sail historical period which we created earlier. A small version of the background image is displayed.

Period	Begin	End	Background image
Age of Sail	1700	1840	

Further historical periods can now be added to the History of Transport timeline. Before doing this, we will create and upload a blank image file to the server. This will act as a placeholder on the shaded graphics background. The program expects an image to be displayed, and an error could occur if a picture image has not yet been uploaded.

Create a grey rectangle with a width of approximately 350 pixels and a height of 300 pixels.



Save this as a graphics file **blank.jpg** and copy it to the **uploads** folder on the server.

We can now add the next historical period. This will have the title '**Steam**', and will cover the years from 1840 to 1950. The scale should be set to **20** pixels/Year.

Click the 'add another historical period' button to load the addPeriod page. You will see that details of the previous historical period are still shown in the input boxes:

History of Transport

Add a historical period:

Period title

Begins in year

Ends in year

The input boxes should be cleared when a new period is to be entered. Open the **timelineDesign.php** file and add a block of code at the start of the program to do this.

```

<?
  session_start();
  $_SESSION["startYear"]='';
  $_SESSION["finishYear"]='';
  $_SESSION["periodTitle"]='';
?>

<html>
<head>
<title> Historical timeline </title>
```

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

Click the 'back' button on the web browser to return to the timelineDesign page. Refresh the page, then select the 'add another historical period' button again. The input boxes should now be blank.

Enter the title **Steam**, and year range from **1840 to 1950**.

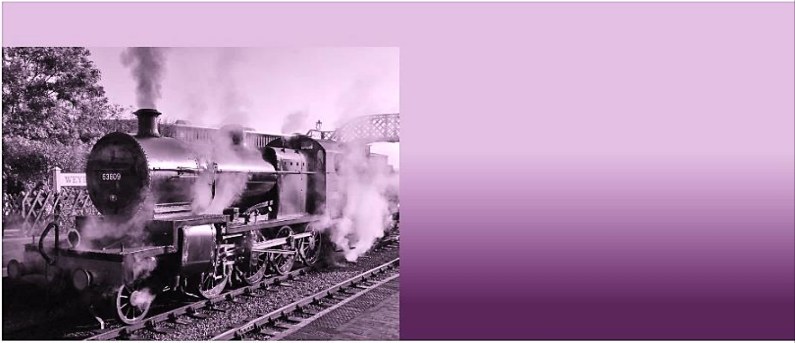
Obtain a picture of a railway locomotive and create a background image. Save this to your computer, then return to the data entry page. Select the saved background, and check that the scale is set to 20 pixels/year. Click the Finished button.

Select background image

Step 1: Select an image filename, then click 'upload image' to add this image to the background.
 No file chosen

Step 2: Click the 'save' button then wait a few seconds as the computer prepares to save the background.

Step 3: Click the 'finished' button to return to the Add Historical Period page and upload your saved background file.



Use the same procedure to add a third historical period:

- **The Modern Era**, from **1950 to 2020**. Set the scale to 30 pixels/Year.

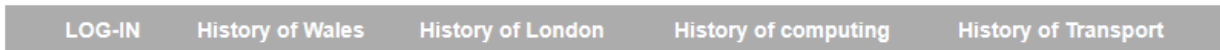
Select a suitable background colour and image.

After the historical period is added, the program will return to the Timeline Design web page and details should be displayed in the table.

Period	Begin	End	Background image
Age of Sail	1700	1840	
Steam	1840	1950	
The Modern Era	1950	2020	

Now that a complete timeline has been designed, we can create a page to display this.

The web site menu will allow the selection of timelines covering different topics, for example:



In order to test this function, use the 'Set up new timeline' option to create several more timelines with different titles. There is no need at this stage to add historical periods to these timelines.

The timelines will be displayed on the index.php page, which loads by default when a user enters the web site URL. Open the **index.php** file and add three blocks of code to the **<body>** section as shown below.

- The first block loads the set of Timeline objects, so that the timeline titles and ID values are available. The first timeline ID found is used as a default, but is replaced if the user selects a different timeline from the menu bar.
- The second block uses a loop to display each of the timeline titles on the menu bar. Clicking a menu option will reload the page, with the selected timeline ID value included in the URL.
- The third block is for test purposes only, to check that the correct set of historical periods is loaded when a timeline is selected from the menu bar.

```

<body>
<?
    include('Timeline.php');
    $timelineCount=Timeline::loadTimelines();
    $timelineWanted = Timeline::$timelines[1]->getTimelineID();
    $timelineID=$_REQUEST['timelineWanted'];
    if ($timelineID>0)
        $timelineWanted = $timelineID;

    echo"<table class=menu>";
    echo"<tr><th class=menu></th>";
    echo"<th class=menu>";
    if ($login=='member')
    {
        echo"<a href='memberArticles.php'>";
        echo"MEMBER PAGE</a></th>";
    }

```

```

else if ($login=='staff')
{
    echo"<a href='staffListArticles.php'>";
    echo"STAFF PAGE</a></th>";
}
else
{
    echo"<a href='login.php'>";
    echo"LOG-IN</a></th>";
}

for ($i=1;$i<=$timelineCount;$i++)
{
    echo"<th class=menu>";
    $timelineID = Timeline::$timelines[$i]->getTimelineID();
    echo"<a href='index.php?timelineWanted=".$timelineID."'>";
    echo Timeline::$timelines[$i]->getTimelineTitle()."</a></th>";
}

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

<?
    include ('TimelinePeriod.php');
    $periodCount = TimelinePeriod::loadByTimelineID($timelineWanted);
    for ($i=1; $i<=$periodCount; $i++)
    {
        echo"<p>".TimelinePeriod::$periods[$i]->getPeriodName();
    }
?>

</body>
</html>

```

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

Run the web site. Select timeline titles from the main menu bar. When the History of Transport timeline is chosen, the titles of the three periods should be displayed.

LOG-IN	History of London	History of computing	History of Transport
Age of Sail			
Steam			
The Modern Era			

We will now work on the graphical display for the timeline. This will be interactive, so **p5.js** will be a suitable programming language.

Re-open the **index.php** file and add links to p5.js in the <head> section.

```

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

<script src="p5.js"></script>
<script src="p5.dom.js"></script>

</head>

```

We will begin by displaying the background image for the first historical period of the History of Transport timeline. Go to the **<body>** section of the **index.php** file. Locate the loop after the table closes which outputs the names of the historical periods. Replace the 'echo' command with a command to load background image names, as shown below. Add a **<script>** block containing JavaScript and p5.js code.



```

</table>
<?
    include ('TimelinePeriod.php');
    $periodCount = TimelinePeriod::loadByTimelineID($timelineWanted);
    for ($i=1; $i<=$periodCount; $i++)
    {
        $Abackground[$i]= TimelinePeriod::$periods[$i]->getImageName();
    }
?>

```

```

<script>
var periodCount=<? echo $periodCount ?>;
var Abackground = <?php echo json_encode($Abackground) ?>;
let imgbk= [];
function preload()
{
    for (i=1;i<=periodCount;i++ )
    {
        filename= Abackground[i];
        imgbk[i]=loadImage('backgrounds/'+filename);
    }
}
function setup()
{
    createCanvas(1200,540);
    background(210);
}
function draw()
{
    background(210);
    drawTimeline();
}
function drawTimeline()
{
    image(imgbk[1],0,0);
}
</script>

```

```

</body>

```

Save the **index.php** file and copy it to the server. Refresh the web page and check that the 'Age of Sail' background image is displayed, with a grey rectangular block below it.

A series of functions are included in the **<script>** block above. **Preload()** ensures that resources are loaded before the web page is run, to avoid displaying incomplete graphics. **Setup()** runs once to create the graphics drawing area on the page. **Draw()** is then run multiple times each second, and calls the **drawTimeline()** function to display the 'Age of Sail' background image. We can add further program code to any of these functions as necessary.

Return to the **index.php** file. To create the timeline display, further data will be needed for each of the historical periods. Add lines of program code to the **<body>** section as shown below. A loop loads the attributes for each timelinePeriod object and stores these in PHP arrays. The PHP arrays are then converted to JavaScript arrays by means of JSON (JavaScript Object Notation) encoding.

```
<?
include ('TimelinePeriod.php');
$periodCount = TimelinePeriod::loadByTimelineID($timelineWanted);
for ($i=1; $i<=$periodCount; $i++)
{
    $Abackground[$i]= TimelinePeriod::$periods[$i]->getImageName();
    $AperiodName[$i]= TimelinePeriod::$periods[$i]->getPeriodName();
    $AperiodStartYear[$i]= TimelinePeriod::$periods[$i]->getStartYear();
    $AperiodFinishYear[$i]= TimelinePeriod::$periods[$i]->getFinishYear();
    $AyearPixels[$i]= TimelinePeriod::$periods[$i]->getYearPixels();
}
?>
<script>
var periodCount=<? echo $periodCount ?>;
var Abackground = <?php echo json_encode($Abackground) ?>;
var AperiodName = <?php echo json_encode($AperiodName) ?>;
var AperiodStartYear = <?php echo json_encode($AperiodStartYear) ?>;
var AperiodFinishYear = <?php echo json_encode($AperiodFinishYear) ?>;
var AyearPixels = <?php echo json_encode($AyearPixels) ?>;

let imgbk= [];
function preload()
{
```

Now that a full set of information about the historical periods is available, we will make use of this to display the title and year range for the 'Age of Sail' period.

Add lines of program code to the **drawTimeline()** function.

```
function drawTimeline()
{
    image(imgbk[1],0,0);
    textSize(20);
    fill(255);
    noStroke();
    text(AperiodName[1],20,30);
    textSize(16);
    text(AperiodStartYear[1]+ " - "+AperiodFinishYear[1],20,52);
}
```

Save the **index.php** file and copy it to the server. Refresh the web page and check that the title and year range are displayed correctly above the picture image.



Return to the **index.php** file. Add lines of program code which will obtain the start year and finish year for the current timeline. These values are then transferred from PHP variables to JavaScript variables.

```

for ($i=1; $i<=$periodCount; $i++)
{
    $Abackground[$i]= TimelinePeriod::$periods[$i]->getImageName();
    $AperiodName[$i]= TimelinePeriod::$periods[$i]->getPeriodName();
    $AperiodStartYear[$i]= TimelinePeriod::$periods[$i]->getStartYear();
    $AperiodFinishYear[$i]= TimelinePeriod::$periods[$i]->getFinishYear();
    $AyearPixels[$i]= TimelinePeriod::$periods[$i]->getYearPixels();
}

for ($i=1; $i<=$timelineCount; $i++)
{
    $timelineID=Timeline::$timelines[$i]->getTimelineID();
    if ($timelineID==$timelineWanted)
    {
        $timelineStart = Timeline::$timelines[$i]->getStartYear();
        $timelineFinish = Timeline::$timelines[$i]->getFinishYear();
    }
}
?>
<script>
var timelineStart=<? echo $timelineStart ?>;
var timelineFinish=<? echo $timelineFinish ?>;
var xoffset=0;
var xoffset2=0;
var down=false;

var periodCount=<? echo $periodCount ?>;
var Abackground = <?php echo json_encode($Abackground) ?>;

```

As a first step towards producing time scale graduations below the background image, we will calculate how many pixels represent the whole timeline from the start year to the finish year. This will depend on the pixel scales used for the different periods.

Add further lines of program code to the **drawTimeline()** method. A loop obtains the number of years and the pixel scale for each of the timeline periods, then adds the required number of pixels to the overall total for the complete timeline.

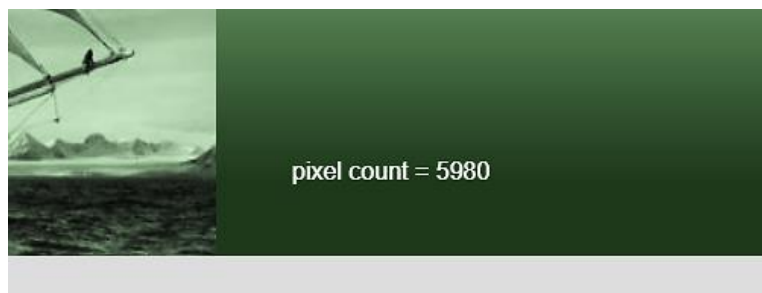

```

text(AperiodName[1],20,30);
textSize(16);
text(AperiodStartYear[1]+ " - "+AperiodFinishYear[1],20,52);

pixelCount=0;
for (i=1;i<=periodCount;i++ )
{
    start=AperiodStartYear[i];
    if (i==1)
        start=timelineStart;
    stop=AperiodFinishYear[i];
    if (i==periodCount)
        stop=timelineFinish;
    extraPixels = (stop-start)* AyearPixels[i];
    pixelCount=pixelCount+extraPixels;
}
text("pixel count = "+pixelCount,500,400);
}
</script>

```

A test output line has been added. Save **index.php** and copy it to the server. Refresh the web page and check that a reasonable number of pixels has been calculated for the History of Transport timeline.



We can now create the time scale. Return to **index.php** and replace the text line outputting the pixel count with the block of program code shown in the two boxes below.

A loop operates for each year of the timescale, from the start year to the finish year. As each year is added, the relevant timeline period is found and the corresponding number of pixels are added. A graduation mark is drawn on the scale, with the year number displayed for each fifth year.

```

    if (i==periodCount)
        stop=timelineFinish;
    extraPixels = (stop-start)* AyearPixels[i];
    pixelCount=pixelCount+extraPixels;
}

xpos = 0;
year = timelineStart;
currentPeriod=1;
while (xpos<= pixelCount)
{
    for(j=2;j<=periodCount;j++ )
    {
        if (year> AperiodStartYear[j])
        {
            currentPeriod=j;
        }
    }
    extra = int(AyearPixels[currentPeriod]);
}

```

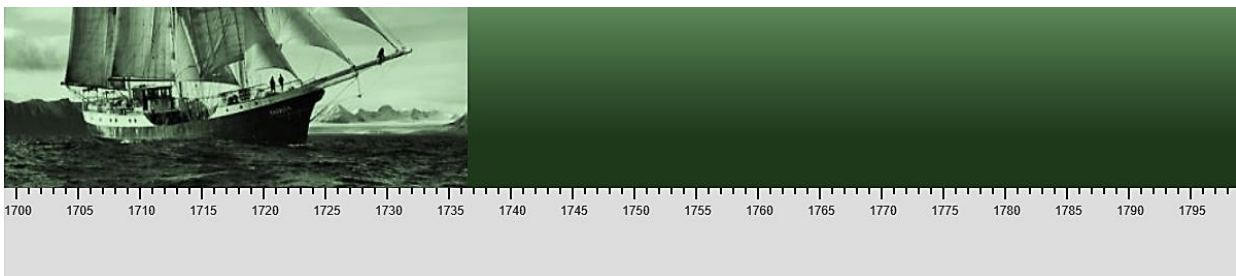
```

        extra = int(AyearPixels[currentPeriod]);

        xpos = xpos + extra;
        stroke(0);
        line(xpos+xoffset+xoffset2,450,xpos+xoffset+xoffset2,455);
        if(year%5== 0)
        {
            textSize(12);
            stroke(0);
            line(xpos+xoffset+xoffset2,450,xpos+xoffset+xoffset2,460);
            fill(0);
            stroke(210);
            text(year,xpos+xoffset+xoffset2-12,476);
        }
        year++;
    }
</script>

```

Save the **index.php** file and copy it to the server. Refresh the web page and check that a timescale is now displayed, beginning at the year 1700 when the History of Transport timeline begins.



Return to the **index.php** file. The next step is to allow the timeline to scroll by dragging the mouse on the time scale. Add lines of program code to the **drawTimeline()** function as shown below.

```

        text(year,xpos+xoffset+xoffset2-12,476);
    }
    year++;
}

textSize(14);
fill(0);
stroke(210);
text('Drag the mouse here to scroll the timeline',600,512);
y=mouseY;
if (y>440)
{
    if (down==true)
        xoffset = x - xstart;
    if (mouseIsPressed)
    {
        x=mouseX;
        if (down==false)
        {
            down=true;
            xstart = mouseX;
        }
    }
}
else

```

```

else
{
    if (down==true)
    {
        down=false;
        xoffset2=xoffset2+xoffset;
        xoffset=0;
    }
}
</script>

```

Save **index.php** and copy it to the server. Refresh the web page, then drag the mouse to the left or right at the point indicated below the timescale. The time scale should now scroll. Notice the difference in the year spacing for the different time periods.



When the web page is first loaded, the offset of the time scale relative to the left edge of the screen is set to zero. This is recorded as the variable **offset2**. By varying this value, the time scale will be offset horizontally and a different range of years will be shown.

When the mouse button is first pressed, the horizontal mouse position is recorded as **xstart**. When the button is released at the end of the dragging operation, the final mouse position is recorded as **x**. The difference between these values gives the distance in pixels that the mouse has moved during the dragging operation, which we record as the variable **offset**. This distance is then applied to the total offset, and the time scale is redrawn in its new horizontal position.

The final step in creating the scrolling timeline is to change the background images as different time periods are dragged across the screen. To do this, return to the **index.php** file and add the block of code below to the **drawTimeline()** function.

```

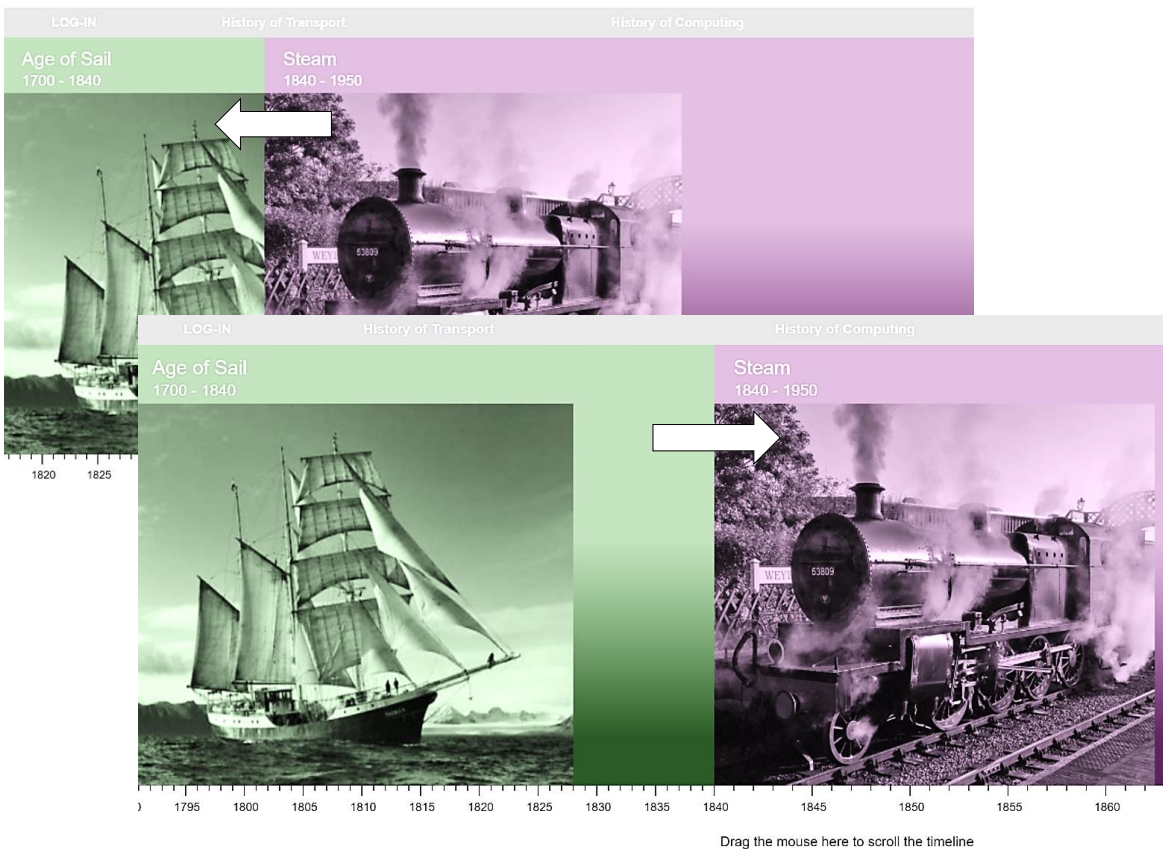
    line(xpos+xoffset+xoffset2,450,xpos+xoffset+xoffset2,460);
    fill(0);
    stroke(210);
    text(year,xpos+xoffset+xoffset2-12,476);
}
for(j=2;j<=periodCount;j++ )
{
    if (year==AperiodStartYear[j])
    {
        loc=xpos+xoffset+xoffset2;
        if (loc<0)
            loc=0;
        image(imgbk[j],loc,0);
        textSize(20);
        fill(255);
        noStroke();
        text(AperiodName[j],loc+20,30);
        textSize(16);
        text(AperiodStartYear[j]+ " - "+AperiodFinishYear[j],loc+20,52);
    }
}
year++;
}

```

The section of program to control the scrolling of the timescale is quite complex, so it has been illustrated in the flowchart below:



Save **index.php** and copy it to the server. Refresh the web page, then drag the mouse to the left or right at the point indicated below the timescale. The time scale should now scroll, with background images moving in and out of the screen display.



We will now leave the timeline display temporarily and work on the entry of historical events.

Open a blank file and add the lines of program code shown on the next page. Save the file as **addevent.php** and copy it to the server.

Run the web site and log-in as staff. Go to STAFF PAGE and select the 'Add event' option. A drop-down list of timelines should appear.

View timeline	Set up new timeline	Edit timeline periods
Add timeline event		
Select timeline:	<div style="border: 1px solid black; padding: 2px;"> History of Transport ▼ History of Wales History of London History of computing History of Transport </div>	

The **<body>** section of the web page begins by loading records from the database to create a set of Timeline objects. A loop then creates arrays of the **timelineID** and **timelineTitle** values, which are used to produce the drop down list.

```

<?
  $r=$_REQUEST['timelineWanted'];
  if ($r>0)
  {
    $timelineWanted=$r;
    $categoryWanted=$_REQUEST['categoryWanted'];
  }
?>
<html>
<head>
  <title> Historical timeline </title>
  <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
<?
  include('staffMenu.php');
  include('Timeline.php');
  $count= Timeline::loadTimelines();
  for ($i=1; $i<=$count;$i++)
  {
    $timelineID[$i]=Timeline::$timelines[$i]->getTimelineID();
    $timelineTitle[$i]=Timeline::$timelines[$i]->getTimelineTitle();
  }
?>
<h4>Add timeline event</h4>
<table>
<tr><td height=30>
  Select timeline:</td>
  <td>
    <?
      echo"<select name='timelineWanted' id='timelineWanted'
      onChange=timelineSelection()>";
      echo"<option>";
      for ($k=1; $k<=$count;$k++)
      {
        if ($timelineWanted==$timelineID[$k])
        {
          $timelineIDwanted=$timelineID[$k];
          echo"<option value='".$timelineID[$k]."' selected>".
          $timelineTitle[$k];
        }
        else
          echo"<option value='".$timelineID[$k]."'>".$timelineTitle[$k];
      }
      echo"</select></td></tr> ";
    ?>
  </td>
</tr>
</table>
</body>
</html>

```

Go to the end of the **addevent.php** page file and add a **timelineSelection()** JavaScript function. This is activated when a timeline is selected from the drop-down list. The web page is then re-loaded, carrying the **timelineID** value as part of the page URL.

```

        echo"</select></td></tr> ";
    ?>
</table>

<script>
function timelineSelection()
{
    timelineWanted = document.getElementById("timelineWanted").value;
    window.location.href='addevent.php?timelineWanted='+timelineWanted;
}
</script>

</body>
</html>

```

The next step is to display a drop-down list of the event categories for the selected timeline. Insert a block of code near the start of the **addevent.php** file to load the category data.

```

if ($r>0)
{
    $timelineWanted=$r;
    $categoryWanted=$_REQUEST['categoryWanted'];

    include('TimelineCategory.php');
    $categoryCount = TimelineCategory::loadByTimelineID($timelineWanted);
    for ($i=1;$i<=$categoryCount;$i++)
    {
        $AcategoryName[$i]=TimelineCategory::$categories[$i]->getCategoryName();
        $Asymbol[$i]=TimelineCategory::$categories[$i]->getSymbol();
    }
}
?>

```

Go now to the **<table>** block in the **<body>** section. Add another drop-down list component as shown below.

```

        echo"<option value='".$timelineID[$k]."'>".$timelineTitle[$k];
    }
    echo"</select></td></tr> ";
    ?>

<tr><td height=30>Category</td>
<?
    echo"<td><select name=categoryWanted id=categoryWanted >";
    echo"<option>";
    for ($h=1;$h<=$categoryCount;$h++)
    {
        if ($categoryWanted==$Asymbol[$h])
        {
            echo"<option value='".$Asymbol[$h]."' selected>".
                $AcategoryName[$h];
        }
        else
        {
            echo"<option value='".$Asymbol[$h]."' >".$AcategoryName[$h];
        }
    }
    echo"</select></td></tr>";
    ?>

</table>

```

Save the **addevent.php** file and copy it to the server. Refresh the web page and select the **History of Transport** timeline from the first drop-down list. Check that the correct set of categories is displayed.

Add timeline event

Select timeline:

Category:

- Air
- Road
- Sea
- Rail
- Air

Re-open the **addevent.php** file. Add further input boxes to the **<table>** as shown below.

The historical event can be recorded as occurring in a *particular year* such as the first commercial flight of Concorde, or *over a period of years* such as the construction of the Channel Tunnel. The title and a text description of the event are then added.

```

        echo"<option value='".$Asymbol[$h]."' >".$AcategoryName[$h];
    }
}
echo"</select></td></tr>";
?>
<tr><td height=30>
    Start year</td>
    <?
        echo"<td><input type=text name='startYear' id='startYear'
            value='$startYear'> Finish year (if different) <input type=text
            id='finishYear' name='finishYear' value='$finishYear'></td></tr>";
    ?>
<tr><td height=40>
    Event title</td>
    <?
        echo"<td><textarea cols=70 rows=2 name='eventTitle' id='eventTitle'>";
        echo $eventTitle;
        echo"</textarea></td></tr>";
    ?>
<tr><td valign='top' height=220>
    Description</td>
    <?
        echo"<td><textarea cols=70 rows=12 name='description' id='description'>";
        echo $description;
        echo"</textarea></td></tr>";
    ?>
<tr><td valign='top'> Image</td>
<td><table border="0" cellpadding="10">
<tr><td><input type="file" name="fileToUpload" id="fileToUpload">
<input type="submit" value="Upload Image" name="submit">
<p>
<?
        echo"<img src='uploads/$imageFile' width='500'>";
        echo"<p><input type='hidden' name='imageFile' id='imageFile'
            value= '$imageFile' >";
    ?>
</td></tr>
</table>
<tr><td></td><td><button type="button" onclick="button_handler()">
    Save record</button></td></tr>
</table>

```


Save the **addevent.php** file and copy it to the server. Refresh the web page and check that input boxes are displayed correctly.

The final stage in entering a historical event record is to upload a picture image for display on the timeline. The user will first click the **'Choose file'** button to open a file selection window. They will then click the **'Upload image'** button to transfer the file to the **uploads** folder on the server.

The file upload will be handled by a separate web page. Open a blank file and add the lines of program code shown below. Save the file as **upload2.php** and copy it to the server.

```
<?
  session_start();
  $timelineWanted = $_REQUEST['timelineWanted'];
  $categoryWanted = $_REQUEST['categoryWanted'];
  $startYear = $_REQUEST["startYear"];
  $finishYear=$_REQUEST['finishYear'];
  $eventTitle=$_REQUEST['eventTitle'];
  $description=$_REQUEST['description'];
  $_SESSION["imagefile"]= $imageFile;
  $_SESSION["startYear"] = $startYear;
  $_SESSION["finishYear"]= $finishYear;
  $_SESSION["eventTitle"]= $eventTitle;
  $_SESSION["description"]= $description;
  $target_dir = "uploads/";
  $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
  $imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
  move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file);
  $imageFile = basename( $_FILES["fileToUpload"]["name"]);
  header("Location: addevent.php?imageFile=$imageFile&timelineWanted=
      $timelineWanted&categoryWanted=$categoryWanted");
?>
```

Re-open the **addevent.php** file. Add a PHP **<form>** command before the **<table>** begins.

```
<h4>Add timeline event</h4>
<?
  echo"<form action='upload2.php?timelineWanted=".$timelineWanted.
    "' name='itemEntryForm' method='post' enctype='multipart/form-data'>";
?>
<table>
```

Add a command to end the form after the final table is closed.

```

        </table>
        <tr><td><td><button type="button" onclick="button_handler()">
                                                    Save record</button>
        </table>
    </form>
    <script>
        function timelineSelection()
    
```

Save the **addevent.php** file and copy it to the server. Refresh the web page and enter a small amount of sample text in the input boxes. Click the 'Choose file' button and select a picture image (.jpg or .png). Now click the 'Upload image' button. Check that the picture file has been uploaded to the **uploads** sub-folder on the server.

You will notice that the picture and test data entered earlier in the input boxes has not been re-loaded. Add the lines of program code shown below to do this. Begin by adding a 'session_start' command to **addevent.php** to allow session variables to be accessed. This must be included as the first line of the program.

```

<?
    session_start();
    $r=$_REQUEST['timelineWanted'];
    if ($r>0)
    {
        $timelineWanted=$r;
        $categoryWanted=$_REQUEST['categoryWanted'];
    }

```

Commands can then be inserted to collect data that was stored previously as session variables. Add the lines of program code below. Save the **addevent.php** file and copy it to the server. Refresh the page and the text should now be displayed.

```


    include('TimelineCategory.php');
    $categoryCount = TimelineCategory::loadByTimelineID($timelineWanted);
    for ($i=1;$i<=$categoryCount;$i++)
    {
        $AcategoryName[$i]=TimelineCategory::$categories[$i]->getCategoryName();
        $Asymbol[$i]=TimelineCategory::$categories[$i]->getSymbol();
    }
    $imageFile=$_REQUEST['imageFile'];
    $startYear = $_SESSION["startYear"] ;
    $finishYear = $_SESSION["finishYear"] ;
    $eventTitle = $_SESSION["eventTitle"] ;
    $description = $_SESSION["description"] ;
}
?>
<html>
<head>
<title> Historical timeline </title>

```

A database table will be required to store timeline event records.

Go to the PHP MyAdmin web page, list the tables in the database, and select the 'new' option. Set up a table with the name **timelineEvent** and add the fields shown below.

The **eventID** field should be specified as the primary key and set to auto-increment as records are added. The maximum size of the file name which can be stored in the **image** field has been set to 30 characters.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	eventID 	int(11)			No	None		AUTO_INCREMENT
2	timelineID	int(11)			No	None		
3	startYear	int(11)			No	None		
4	finishYear	int(11)			Yes	NULL		
5	eventTitle	text	latin1_swedish_ci		No			
6	category	int(11)			No	None		
7	eventText	text	latin1_swedish_ci		No			
8	image	varchar(30)	latin1_swedish_ci		No	None		
9	display	int(11)			No	0		
10	submissionDate	varchar(16)	latin1_swedish_ci		No	None		
11	contributor	varchar(30)	latin1_swedish_ci		No	None		

A **TimelineEvent** class can now be created to handle database operations. Open a blank file and add the lines of program code in the two boxes below. The attributes of a TimelineEvent object are defined, then a constructor method is added. We have also included a method to store event records into the database.

```

<?
class TimelineEvent
{
    public static $events=array();
    private $eventID;
    private $timelineID;
    private $startYear;
    private $finishYear;
    private $eventTitle;
    private $category;
    private $eventText;
    private $image;
    private $display;
    private $submissionDate;
    private $contributor;

    function __construct($eventID,$timelineID,$startYear,$finishYear,
                        $eventTitle, $category,$eventText,$image,
                        $display,$submissionDate,$contributor)
    {
        $this->eventID = $eventID;
        $this->timelineID = $timelineID;
        $this->startYear = $startYear;
        $this->finishYear = $finishYear;
        $this->eventTitle = $eventTitle;
        $this->category = $category;
        $this->eventText = $eventText;
        $this->image = $image;
        $this->display = $display;
        $this->submissionDate = $submissionDate;
        $this->contributor = $contributor;
    }
}

```

```

public static function addEvent($timelineID,$startYear,$finishYear,
                               $eventTitle, $category,$eventText,$image, $display,
                               $submissionDate,$contributor)
{
    include('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="INSERT INTO timelineEvent VALUES ( ' ', '$timelineID',
        '$startYear', '$finishYear', '$eventTitle', '$category', '$eventText',
        '$image', '$display', '$submissionDate', '$contributor')";
    $result=mysqli_query($conn, $query);
    mysqli_close($conn);
}
}
?>

```

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

We can now use the **addEvent()** method to save an event record into the database table. Return to the **addevent.php** file and add a Javascript function which will be called when the 'Save record' button is clicked. Save the **addevent.php** file and copy it to the server.

```

<script>
function timelineSelection()
{
    timelineWanted = document.getElementById("timelineWanted").value;
    window.location.href='addevent.php?timelineWanted='+timelineWanted;
}

function button_handler()
{
    startYear = document.getElementById("startYear").value;
    finishYear = document.getElementById("finishYear").value;
    eventTitle = document.getElementById("eventTitle").value;
    category = document.getElementById("categoryWanted").value;
    description = document.getElementById("description").value;
    imageFile = document.getElementById("imageFile").value;
    var error=false;
    var n = eventTitle.length;
    if (n<2)
    {
        alert("An event title must be entered");
        error=true;
    }
    if (error==false)
    {
        timelineWanted = document.getElementById("timelineWanted").value;
        destination = 'saveEvent.php?startYear='+startYear+'&finishYear='+
            finishYear+'&category='+ category+'&eventTitle='+ eventTitle +
            '&description=' + description+'&imagefile=' + imageFile +
            '&timelineWanted=' + timelineWanted;
        window.location.href= destination;
    }
}
</script>
</body>
</html>

```

The function begins by carrying out a presence check on the Title field, to ensure that this has not been left blank. Similar validation could be carried out for other fields if required. If no error is detected, the program will proceed to another web page where the record will be saved. The field values are carried as attachments to the URL web page address.

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

```

<?
    session_start();
    $startYear=$_REQUEST['startYear'];
    $finishYear=$_REQUEST['finishYear'];
    $eventTitle=$_REQUEST['eventTitle'];
    $description=$_REQUEST['description'];
    $imageFile=$_REQUEST['imagefile'];
    $categoryWanted=$_REQUEST['category'];
    $timelineWanted=$_REQUEST['timelineWanted'];
    $userWanted=$_SESSION['user'];
    $status=$_SESSION['login'];
    if ($status=='member')
    {
        $displayCode=1;
    }
    else
    {
        $displayCode=0;
    }
    $date = date('d/m/Y');
    $eventTitle = str_replace("'", "`", $eventTitle);
    $description = str_replace("'", "`", $description);
?>
<html>
<head>
<title> Historical timeline </title>
</head>
<body>
<?
    include('TimelineEvent.php');
    TimelineEvent::addEvent($timelineWanted,$startYear,$finishYear,
        $eventTitle,$categoryWanted,$description, $imageFile,
        $displayCode,$date,$userWanted);
    header('Location: index.php');
?>
</body>
</html>

```

The page begins by collecting most of the data fields required for the timeline event record. The **status** field is set according to whether the record was entered by staff and can be immediately displayed, or whether it was entered by society member and needs to be approved by staff before display.

Run the website, logging-in as staff and go to STAFF PAGE. Enter a complete event record for the History of Transport timeline, such as the first commercial flight of Concorde in 1978, as in the example below. Click the 'Save record' button.

Select timeline:

Category:


Start year: Finish year (if different):

Event title:

Description:

```
On 21st January 1976, at 11:40am, Concorde made its first commercial flight, unbeknown that only 27 years later it would be pulled from passenger service. In the short amount of time, Concorde made a large, and lasting impact on travellers who would be reminiscing about their flight (or inability to get a flight) on the iconic aircraft. That first flight took passengers from London Heathrow to Bahrain, and from Paris to Rio de Janeiro. The appeal to frequent flyers being that flying on this high-speed aircraft would minimise your flight time by up to two-thirds. The fastest ever crossing from New York to London was completed in only 2 hours, 52 minutes and 59 seconds; hardly enough time to have a meal and watch a movie, but perfect for those business passengers. However, having this luxury of a minuscule flight time meant a costly bill, costing
```

Image: No file chosen



Go to the PHP MyAdmin web page and open the timelineEvent table. Check that the record has been saved correctly.

Return to the historical timeline web site and enter more events for the History of Transport timeline. It will be important to check that the timeline can correctly display more than one event for the same year. With this in mind, create test data for two separate events with the same year date.

We can now return to the timeline to display the historical events. However, before doing this it will be necessary to add methods to the TimelineEvent class. We will need to reload the event records and create a set of TimelineEvent objects, and then access the attributes of these objects.

Re-open the **TimelineEvent.php** class file. Add the **loadByTimelineID()** and **get()** methods shown below.

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

```

public static function loadByTimelineID($timelineIDwanted)
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM timelineEvent WHERE timelineID = ".
        $timelineIDwanted." ORDER BY startYear";
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    $i=1;
    while ($i <= $num)
    {
        $row=mysqli_fetch_assoc($result);
        $eventID=$row["eventID"];
        $timelineID=$row["timelineID"];
        $startYear=$row["startYear"];
        $finishYear=$row["finishYear"];
        $eventTitle=$row["eventTitle"];
        $category=$row["category"];
        $eventText=$row["eventText"];
        $image=$row["image"];
        $display=$row["display"];
        $submissionDate=$row["submissionDate"];
        $contributor=$row["contributor"];
        TimelineEvent::$events[$i] = new TimelineEvent($eventID,
            $timelineID,$startYear,$finishYear,$eventTitle,$category,
            $eventText,$image,$display,$submissionDate,$contributor);
        $i++;
    }
    return $num;
}

public function getEventID(){return $this->eventID;}
public function getTimelineID(){return $this->timelineID;}
public function getStartYear(){return $this->startYear;}
public function getFinishYear(){return $this->finishYear;}
public function getEventTitle(){return $this->eventTitle;}
public function getCategory(){return $this->category;}
public function getEventText(){return $this->eventText;}
public function getImage(){return $this->image;}
public function getDisplay(){return $this->display;}
public function getSubmissionDate(){return $this->submissionDate;}
public function getContributor(){return $this->contributor;}

```

```

}
?>

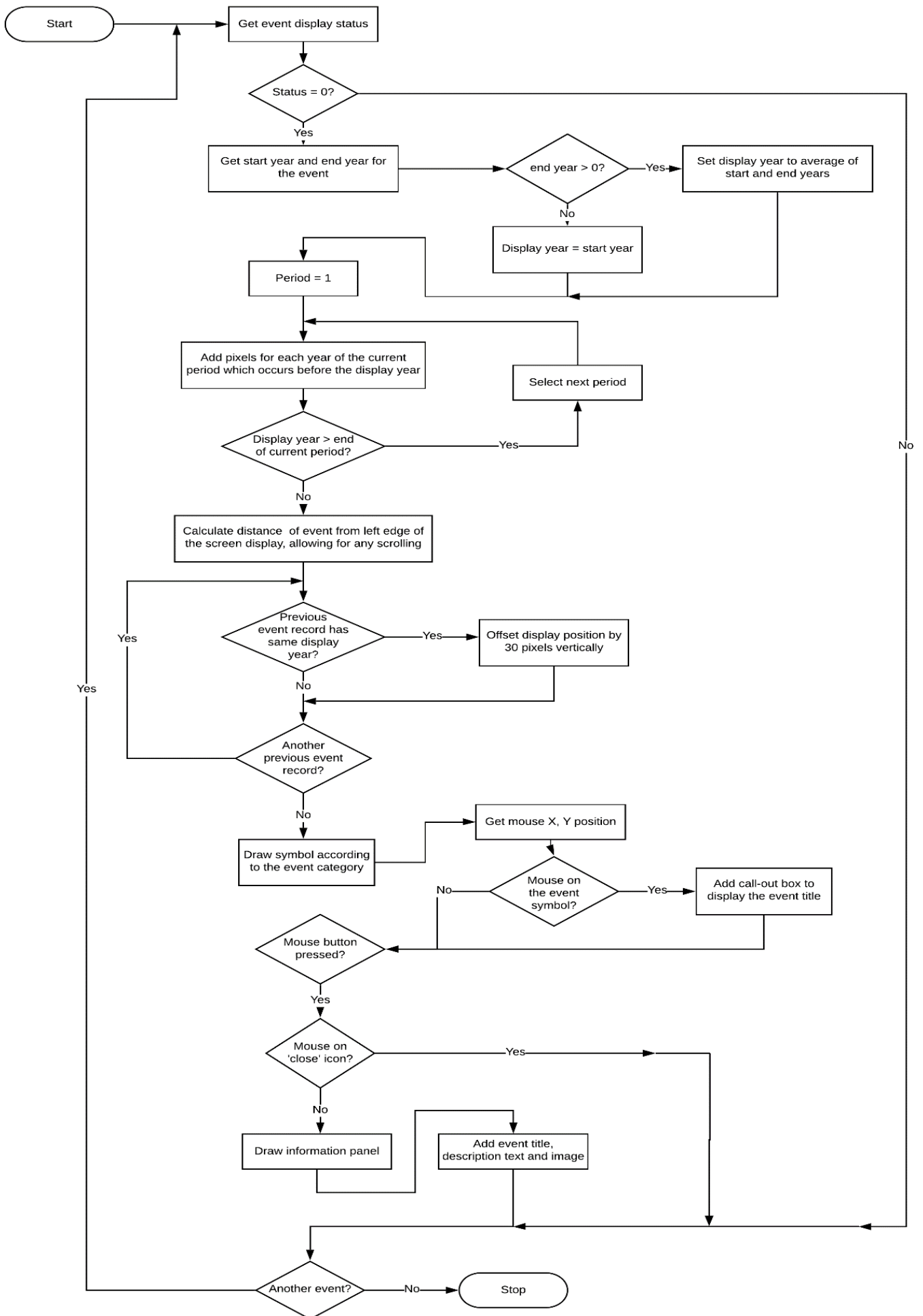
```

The next step is to add program code to the **index.php** file to display historical events. The program sequence is illustrated in the flowchart below.

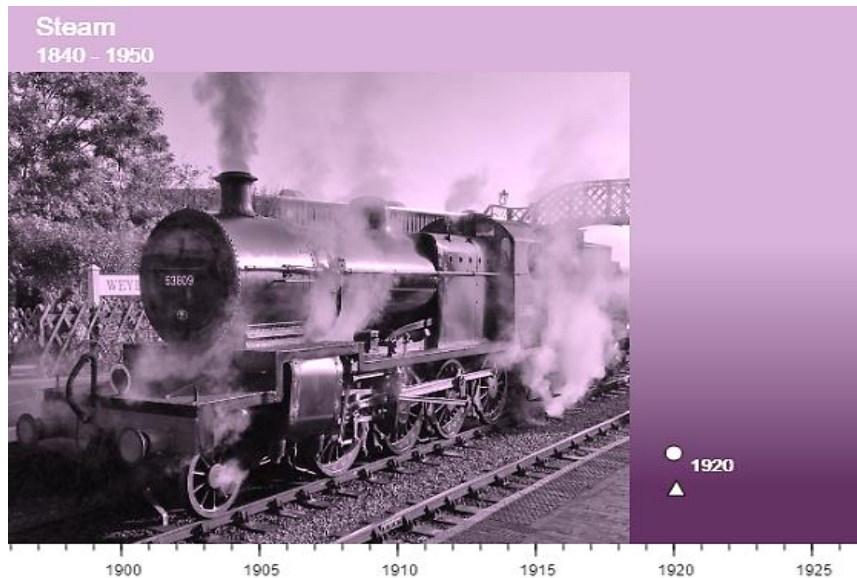
A loop operates for each of the historical events for the selected timeline. Only the events which have been approved for display (with status = 0) will be processed.

If only a start year has been entered for the event, this is used as the display year. If both start and finish years have been given, then the display year is taken as the mid-point.

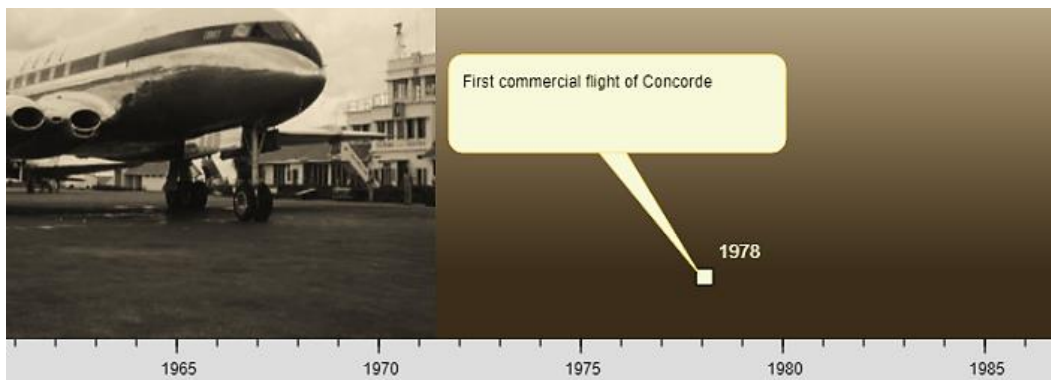
The program works through each of the timeline periods in chronological order. If the display year lies within or after the current period, then the necessary number of pixels are added to the display position. The current timeline scroll position is then applied to this total, so the display position is recalculated relative to the left side of the screen display.



Previously processed events are checked to determine whether any occur in the same year as the current event. If so, the display position of the current event symbol is adjusted so that it will lie vertically above any previous symbols. The symbol is then plotted on the graphical display.



The position of the mouse pointer is then checked. If it is close to an event symbol, a call-out box is drawn and the event title displayed.



If the mouse button is pressed, then two possibilities exist:

- The user has clicked on the event symbol. A display panel should then open.

Ste
184
✕

1920

Delivery of London Underground 1920 Stock

Following the end of the First World War, consideration was given to how improvements to the service on the Underground could be improved. One of the main factors in the slow running of the service was the time taken at stations to close all the doors or gates. In order to address this issue, a batch of 20 trailer and 20 control trailer cars was ordered from Cammell Laird in 1919. These would be the first vehicles to be designed with air-operated doors. In order to make up complete trains of air-operated stock, twenty Gate Stock driving motor cars, built in France in 1906, were reconstructed with air doors, and were normally formed into six-car trains. The cars were the forerunners to the Standard Stock, which became the most prolific group of tube stock to operate on the system, thanks in part to the Trade Facilities Act 1921.

- A second possibility is that a display panel is already open, and the user has clicked on the 'close' icon in the top right hand corner. In this case, the panel will not be displayed.

Begin by producing a 'close' icon which will be displayed in the top right hand corner of the event information panel. This should be approximately 24 pixels square.



Save the graphic image as 'close.png' and copy it to the timeline folder on the server.

Re-open the **index.php** file. Add lines of program code at the start of the <body> section to include the TimelineEvent and TimelineCategory class files.

```
<body>
<?
include('Timeline.php');
include('TimelineEvent.php');
include('TimelineCategory.php');

$timelineCount=Timeline::loadTimelines();
$timelineWanted = Timeline::$timelines[1]->getTimelineID();
$timelineID=$_REQUEST['timelineWanted'];
```

Add program code as shown in the first box below. This will load the event and category records for the current timeline and create sets of objects. Loops in the program then access each of the objects by means of **get()** methods to obtain the attribute values. These values are then stored in a series of arrays.

```
$timelineWanted = Timeline::$timelines[1]->getTimelineID();
$timelineID=$_REQUEST['timelineWanted'];
if ($timelineID>0)
    $timelineWanted = $timelineID;

$eventCount = TimelineEvent::loadByTimelineID($timelineWanted);
for ($i=1;$i<=$eventCount;$i++)
{
    $title = TimelineEvent::$events[$i]->getEventTitle();
    $AsymbolCategory[$i]=TimelineEvent::$events[$i]->getCategory();
    $AdisplayStatus[$i]=TimelineEvent::$events[$i]->getDisplay();
    $AstartYear[$i]=TimelineEvent::$events[$i]->getStartYear();
    $AfinishYear[$i]=TimelineEvent::$events[$i]->getFinishYear();
    $AeventTitle[$i]=TimelineEvent::$events[$i]->getEventTitle();
    $AeventText[$i]=TimelineEvent::$events[$i]->getEventText();
    $Aimage[$i]=TimelineEvent::$events[$i]->getImage();
}
$categoryCount=TimelineCategory::loadByTimelineID($timelineWanted);
for ($i=1;$i<=$categoryCount;$i++)
{
    $AcategoryName[$i]=TimelineCategory::$categories[$i]->getCategoryName();
}

echo"<table class=menu>";
echo"<tr><th class=menu>";
```

Go next to the **<script>** block which follows the PHP program code. Add the series of lines of code in the second box below. These will convert PHP variables to JavaScript variables for use in drawing the timeline graphics. The **JSON encode** function allows PHP arrays to be transferred to JavaScript.

```

var xoffset=0;
var xoffset2=0;
var down=false;

var eventCount=<? echo $eventCount ?>;
var AsymbolCategory=<?php echo json_encode($AsymbolCategory) ?>;
var AdisplayStatus=<?php echo json_encode($AdisplayStatus) ?>;
var AstartYear=<?php echo json_encode($AstartYear) ?>;
var AfinishYear=<?php echo json_encode($AfinishYear) ?>;
let imageYear = [];
let imagePosition = [];
var AeventTitle=<?php echo json_encode($AeventTitle) ?>;
let showInformation= [];
var AeventText=<?php echo json_encode($AeventText) ?>;
var Aimage=<?php echo json_encode($Aimage) ?>;
let picture= [];
var AcategoryName=<?php echo json_encode($AcategoryName) ?>;
var categoryCount=<? echo $categoryCount ?>;

var periodCount=<? echo $periodCount ?>;
var Abackground = <?php echo json_encode($Abackground) ?>;

```

Go to the **<script>** block and locate the **preload()** function. Add lines of code which will download the 'close' icon and the picture images for the current timeline, so that they are immediately available when the program starts.

```

function preload()
{
  for (i=1;i<=periodCount;i++ )
  {
    filename= Abackground[i];
    imgbk[i]=loadImage('backgrounds/'+filename);
  }

  close=loadImage("close.png");
  for (i=1;i<=eventCount;i++ )
  {
    picture[i]=loadImage("uploads/"+Aimage[i]);
  }
}

```

Within the **draw()** function, add a call to a function **showEvents()**. Set up the **showEvents()** function below the **draw()** function.

```

function setup()
{
  createCanvas(1200,540);
  background(210);
}
function draw()
{
  background(210);
  drawTimeline();

  showEvents();
}

function showEvents()
{
}

function drawTimeline()
{

```

Within the **showEvents()** function, create a loop which will operate for each of the historical events in the current timeline. Add a conditional block, which will only operate if the event has a status code of zero, meaning that it has been approved for display on the web page.

```
function showEvents()
{
    x=mouseX;
    y=mouseY;
    for (i=0;i<=eventCount;i++ )
    {
        offset=0;
        fill(245,246,206);
        code=int(AsymbolCategory[i]);
        if (AdisplayStatus[i]==0)
        {
        }
    }
}
```

We now begin the calculation of the event's position along the timeline. If both start and finish years are recorded for the event, the event year is taken as the average of these. The first historical period is then processed. The event will be within or after this period. The appropriate number of years is calculated and converted to a distance in pixels along the timescale. Add the lines of program code shown below.

```
fill(245,246,206);
code=int(AsymbolCategory[i]);
if (AdisplayStatus[i]==0)
{
    year = int(AstartYear[i]);
    if (AfinishYear[i]>0)
    {
        year = int((int(AstartYear[i])+int(AfinishYear[i]))/2);
    }
    extraYears = year-timelineStart;
    lastYear=AperiodFinishYear[1];
    if (year<AperiodFinishYear[1])
    {
        lastYear=year;
    }
    includeYears = int(lastYear-timelineStart);
    xpos = int(includeYears * AyearPixels[1]);
}
```

A loop is then used to check each of the remaining historical periods. If the event lies within or after a period, the number of years to be included is calculated and the appropriate number of pixels added to the position of the event along the timeline. Add the lines of program code below.

```

    if (year<AperiodFinishYear[1])
    {
        lastYear=year;
    }
    includeYears = int(lastYear-timelineStart);
    xpos = int(includeYears * AyearPixels[1]);

    for (j=2;j<=periodCount ;j++ )
    {
        if (year>AperiodStartYear[j])
        {
            lastYear=AperiodFinishYear[j];
            if (year<AperiodFinishYear[j])
            {
                lastYear=year;
            }
            includeYears = int(lastYear-AperiodStartYear[j]);
            xpos = xpos + int(includeYears * AyearPixels[j]);
        }
    }
}

```

We now know the horizontal position of the event along the timeline. It is just necessary to check whether more than one event is to be displayed for the same year; in this case, the event symbols must be offset vertically so that they remain separate. Add the lines of program code below to check for multiple events in a year and apply the offset.

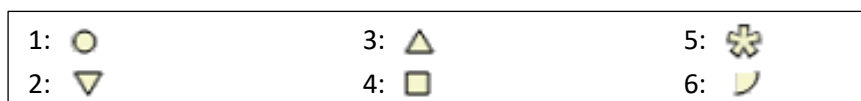
```

        includeYears = int(lastYear-AperiodStartYear[j]);
        xpos = xpos + int(includeYears * AyearPixels[j]);
    }
}

loc=xpos+xoffset+xoffset2+32;
loc=loc-int(20-AyearPixels[1]);
imageYear[i]=year;
var previousCount=0;
for (p=0;p<i;p++)
{
    if (imageYear[p]==imageYear[i])
    {
        previousCount++;
    }
}
imagePosition[i]=previousCount;
var offset=previousCount*30;
}

```

A symbol can now be plotted at the appropriate point on the timeline for the historical event. A set of six geometrical symbols has been selected:



Add lines of program code to display the correct symbol according to the event's category.

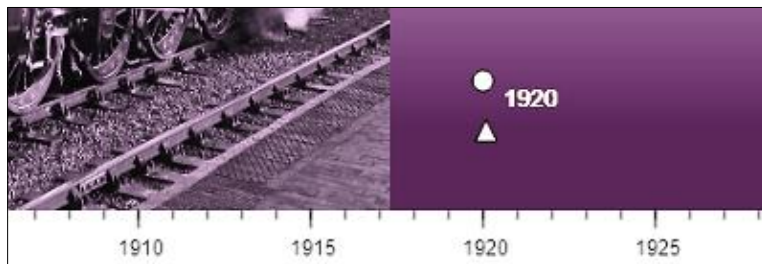
```

imagePosition[i]=previousCount;
var offset=previousCount*30;

stroke(0);
switch (code)
{
  case 1: ellipse(loc-12,410-offset-5,12,12); break;
  case 2: triangle(loc-16,410-offset-12,loc-4,410-offset-12,loc-10,410-offset);
          break;
  case 3: triangle(loc-16,410-offset,loc-4,410-offset,loc-10,410-offset-12);
          break;
  case 4: rect(loc-16,410-offset-12,12,12);break;
  case 5: textSize(44); strokeWeight(2);
          text('*',loc-22,410-offset+20); strokeWeight(1);break;
  case 6: arc(loc-17,410-offset-10,20,20,0,HALF_PI); break;
}
textSize(14);
noStroke();
text(year,loc,390);
}

```

Save the **index.php** file and copy it to the server. Run the website and display the History of Transport timeline. Drag the mouse to scroll the timeline. Symbols and dates should be displayed at the correct positions for the historical events entered. Check that events occurring in the same year are shown as separate symbols.



Re-load the **index.php** file and return to the **showEvents()** function. Add the lines of program code shown in the two boxes below:

```

case 5: textSize(44); strokeWeight(2);
        text('*',loc-22,410-offset+20); strokeWeight(1);break;
case 6: arc(loc-17,410-offset-10,20,20,0,HALF_PI); break;
}
textSize(14);
noStroke();
text(year,loc,390);

if ((y>395-offset)&&(y<415-offset))
{
  if ((x>loc-25)&&(x<loc+5))
  {
    if (loc>600)
    {
      x2=loc-70;
      x3=loc-90;
      fill(245,246,206);
      stroke(250,219,103);
      triangle(loc-14, 410-offset-8, x2, 310-offset, x3, 310-offset);
    }
  }
}

```


When the mouse button is pressed on an event symbol, an information panel will be displayed. Add the lines of program code below to do this.

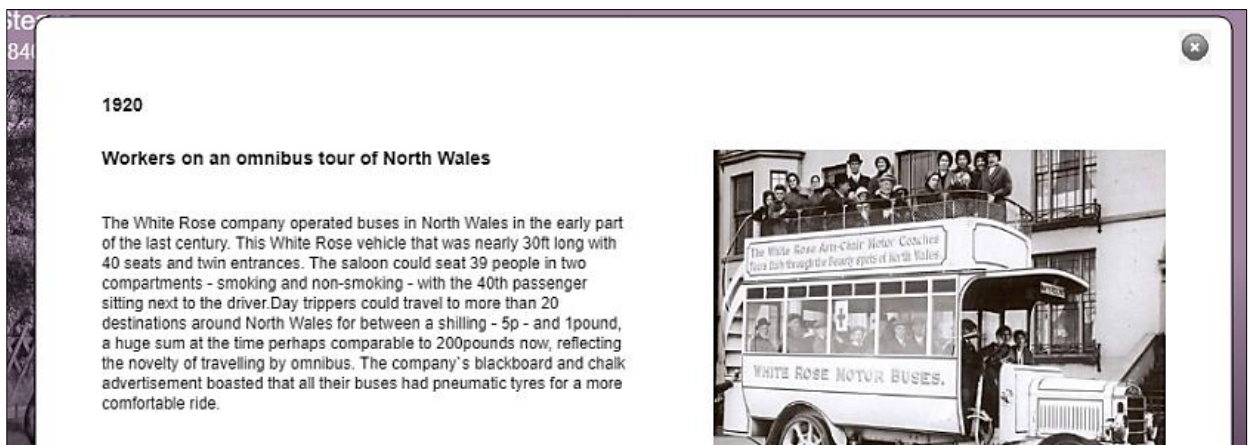
```

        if (mouseIsPressed)
        {
            showInformation[i]=true;
        }
    }
}
}

for (i=1;i<=eventCount;i++ )
{
    if (showInformation[i]==true)
    {
        stroke(0);
        fill(255);
        rect(50,20,900,495,10);
        image(close,910,30);
        fill(0);
        stroke(255);
        textSize(14);
        textStyle(BOLD);
        dateWanted = AstartYear[i];
        if (int(AfinishYear[i])>1000)
        {
            dateWanted = dateWanted + " - " + AfinishYear[i];
        }
        text(dateWanted,100,80,400,200);
        text(AeventTitle[i],100,120,400,200);
        textSize(14);
        textStyle(NORMAL);
        text(AeventText[i],100,170,400,300);
        image(picture[i],560,120,340,280);
    }
}
textStyle(NORMAL);
}
}

```

Save the **index.php** file and copy it to the server. Run the website and display the History of Transport timeline. Scroll the timeline until an event symbol is visible, then click the mouse button on the symbol. The program creates a white panel above the timeline background. The 'close' icon is added in the top corner, as in the illustration below. The year or year range for the event is displayed, along with the event title, descriptive text and illustration.



The event title and text description have been transferred to JavaScript variables using the **JSON encode** function. Whilst this works correctly for alphanumeric characters and standard punctuation, problems may occur with special characters. Errors can be caused by accented letters and keyboard characters including, surprisingly, the £ or — symbol. If text fails to appear, check for special characters and remove or replace them if necessary.

Return to the **index.php** file and add the lines of program code below to the **showEvents()** function.

```

        textSize(14);
        textStyle(NORMAL);
        text(AeventText[i],100,170,400,300);
        image(picture[i],560,120,340,280);
    }
}

if (mouseIsPressed)
{
    x=mouseX;
    y=mouseY;
    if ((x>910)&&(x<936)&&(y>30)&&(y<56))
    {
        for (i=1;i<=eventCount;i++ )
        {
            showInformation[i]=false;
        }
    }
}

textStyle(NORMAL);
}

```

Save the **index.php** file and copy it to the server. Run the website and display an event panel for the History of Transport timeline. Check that the panel closes when the 'close' icon is clicked.

The only task remaining on the timeline display is to provide a key to the event category symbols. Return to the **index.php** file and add the lines of program code in the two boxes below to the **drawTimeline()** function to produce the key.

```

        noStroke();
        text(AperiodName[j],loc+20,30);
        textSize(16);
        text(AperiodStartYear[j]+ " - "+AperiodFinishYear[j],loc+20,52);
    }
}
year++;
}

noStroke();
var up=500;
var across=60;
for (h=1;h<=categoryCount;h++)
{
    fill(0);
    noStroke();
    textSize(12);
    text(AcategoryName[h],across,up);
    fill(245,246,206);
    stroke(0);
    switch (h)
    {
        case 1: ellipse(across-12,up-5,10,10); break;
    }
}

```

```

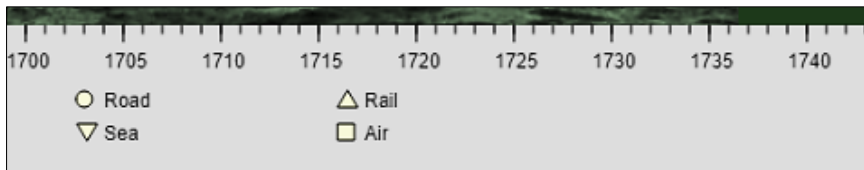
case 1: ellipse(across-12,up-5,10,10); break;

case 2: triangle(across-16,up-10,across-4,up-10,across-10,up); break;
case 3: triangle(across-16,up,across-4,up,across-10,up-10); break;
case 4: rect(across-16,up-10,10,10);break;
case 5: textSize(44); strokeWeight(2);
      text('*',across-22,up+20); strokeWeight(1);break;
case 6: arc(across-17,up-10,20,20,0,HALF_PI); break;
}
up=up+20;
if (up>520)
{
  up=500;
  across=across+160;
}
}

textSize(14);
fill(0);
stroke(210);
text('Drag the mouse here to scroll the timeline',600,512);

```

Save the **index.php** file and copy it to the server. Run the website and display the History of Transport timeline. Check that a correct set of captions are displayed for the category symbols.



This completes the work on the timeline display.

In addition to the staff of the history society, members are also permitted to add events to timelines. We will work on this facility next. Begin by opening a blank file and adding the lines of program code below. These will create a member menu bar which includes an option to add a timeline event.

```

<?
  echo"<table class=menu>";
  echo"<tr><th class=menu>";
  echo"<th class=menu>";
  echo"<a href='index.php'>";
  echo"View timeline</a>";

  echo"<th class=menu>";
  echo"<a href='memberArticles.php'>";
  echo"Member page</a>";

  echo"<th class=menu>";
  echo"<a href='addevent.php'>";
  echo"Add event</a>";

  echo"<th class=menu>";
  echo"<a href='login.php'>";
  echo"LOG OUT</a>";

  echo"</table>";
?>

```

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

Open another blank file and add the lines of program code below. Save the file as **memberArticles.php** and copy it to the server. This creates a member page which can be accessed from the timeline display.

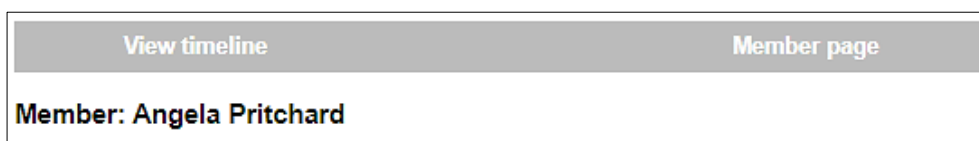
```
<?
    session_start();
    $userWanted=$_SESSION['user'];
?>
<html>
  <head>
    <title> Historical timeline </title>
    <link rel="Stylesheet" type="text/css" href="styleSheet.css" />
  </head>
  <body>
    <?
      include('memberMenu.php');
      include('TimelineLogin.php');
      TimelineLogin::loadTimelineUsers();
      $name=TimelineLogin::nameOfUser($userWanted);
      echo"<h3>Member: ".$name."</h3>";
    ?>
  </body>
</html>
```

The page will display the name of the member. To do this, an additional method must be added to the TimelineLogin class. Open the file **TimelineLogin.php** and add the **nameOfUser()** method shown below.

```
public static function nameOfUser($userWanted)
{
    $answer="";
    for ($i=1;$i<=TimelineLogin::$userCount;$i++)
    {
        if ($userWanted==TimelineLogin::$timelineUser[$i]->user)
        {
            $forename = TimelineLogin::$timelineUser[$i]->forename;
            $surname = TimelineLogin::$timelineUser[$i]->surname;
            $answer = $forename." ".$surname;
        }
    }
    return $answer;
}
?>
```

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

Run the website. Log-in or register as a member. From the menu above the timeline display, select the MEMBER PAGE option. Check that the menu bar and member's name are displayed, as in the example below.



Members will be able to enter historical events using the same input page as the staff. However, the page should display the member menu rather than the staff menu. Open the **addEvent.php** file and replace the **include('staffMenu.php')** command with the block of program code below.

```

<title> Historical timeline </title>
<link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
<?
    $login = $_SESSION['login'];
    if ($login == 'staff')
    {
        include('staffMenu.php');
    }
    else
    {
        include('memberMenu.php');
    }

    include('Timeline.php');
    $count= Timeline::loadTimelines();

```

Save the **addEvent.php** file and copy it to the server. Return to the member web page and select the 'Add event' menu option. The input page should open as before, but now displaying the member menu bar. Add a historical event record for the History of Transport timeline, as in this example:

View timeline	Member page	Add event	LOG OUT
---------------	-------------	-----------	---------

Add timeline event

Select timeline:


Category:

Start year: Finish year (if different):

Event title:

Description:

Image: No file chosen



Click the 'Save record' button. Check that the image selected has been copied to the uploads folder on the server.

Go to the PHP MyAdmin web page and check that the event record has been added correctly to the timelineEvent table in the database. The value for the **display** field should be set to 1, indicating that the record needs to be approved by staff before it is displayed on the timeline.

We will now produce a listing for the member to show the historical events which they have submitted. To do this, an additional method needs to be added to the TimelineEvent class. Open the **TimelineEvent.php** file and add a **loadEvents()** method as shown below. This method will load all timelineEvent records and create a corresponding set of objects.

```
public static function loadEvents()
{
    include ('user.inc');
    $conn = new mysqli(localhost, $username, $password, $database);
    if (!$conn) {die("Connection failed: ".mysqli_connect_error()); }
    $query="SELECT * FROM timelineEvent ORDER BY startYear";
    $result=mysqli_query($conn, $query);
    $num=mysqli_num_rows($result);
    mysqli_close($conn);
    $i=1;
    while ($i <= $num)
    {
        $row=mysqli_fetch_assoc($result);
        $eventID=$row["eventID"];
        $timelineID=$row["timelineID"];
        $startYear=$row["startYear"];
        $finishYear=$row["finishYear"];
        $eventTitle=$row["eventTitle"];
        $category=$row["category"];
        $eventText=$row["eventText"];
        $image=$row["image"];
        $display=$row["display"];
        $submissionDate=$row["submissionDate"];
        $contributor=$row["contributor"];
        TimelineEvent::$events[$i] = new TimelineEvent($eventID,$timelineID,
            $startYear,$finishYear,$eventTitle,$category,$eventText,
            $image,$display,$submissionDate,$contributor);
        $i++;
    }
    return $num;
}
}
?>
```

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

Return to **memberArticles.php** and add the block of program code shown on the next page. This loads the set of timelineEvent objects, then selects the events submitted by the member. Details are displayed in a table, with a message indicating whether the record has been published or is awaiting approval.

Save the updated **memberArticles.php** file and copy it to the server. Run the website and log-in as a member. Go to the member page and check that a table displays the article submitted by the member. Enter several more timeline events for this member. When viewed in the table, these should all be listed as 'awaiting approval' as in the example shown below.

Member: Darren Williams			
Your articles:			
Date	Event	Date submitted	Status
1862	The Flying Scotsman	09/07/2020	awaiting approval
1938	Mallard breaks the world speed record for a steam locomotive	11/07/2020	awaiting approval
1938	Ocean liner Queen Elizabeth is launched	11/07/2020	awaiting approval
1961	E-type Jaguar	05/07/2020	awaiting approval
1969	The era of the Jumbo Jet begins	05/07/2020	awaiting approval

```

<body>
<?
  include('memberMenu.php');
  include('TimelineLogin.php');
  TimelineLogin::loadTimelineUsers();
  $name=TimelineLogin::nameOfUser($userWanted);
  echo"<h3>Member: ".$name."</h3>";
?>
<br>
Your articles:
<br><br>
<table cellpadding=5 >
<?
  echo"<tr><th class = outline>Date</th>";
  echo"<th class = outline>Event</th>";
  echo"<th class = outline>Date submitted</th>";
  echo"<th class = outline>Status</th></tr>";
  include('TimelineEvent.php');
  $count = TimelineEvent::loadEvents();
  for ($i=1; $i<=$count; $i++)
  {
    $contributor = TimelineEvent::$events[$i]->getContributor();
    if ($contributor==$userWanted)
    {
      echo"<tr>";
      echo"<td>".TimelineEvent::$events[$i]->getStartYear()."</th>";
      echo"<td>".TimelineEvent::$events[$i]->getEventTitle()."</th>";
      echo"<td>".TimelineEvent::$events[$i]->getSubmissionDate()."</th>";
      $status = TimelineEvent::$events[$i]->getDisplay();
      if ($status == 0)
        echo"<td>published</th></tr>";
      if ($status == 1)
        echo"<td>awaiting approval</th></tr>";
    }
  }
?>
</table>
</body>
</html>

```

The final task in this project is to provide a facility for staff to view event records submitted by members, and approve these for publication if suitable. Open a blank file and add the lines of program code shown in the two boxes below.

```

<?
  session_start();
  include('TimelineEvent.php');
  $userWanted=$_SESSION['user'];
  $approvedRecords = $_REQUEST['approvedRecords'];
?>
<html>
<head>
  <title> Historical timeline </title>
  <link rel="stylesheet" type="text/css" href="styleSheet.css" />
</head>
<body>
<?
  include('staffMenu.php');
  include('TimelineLogin.php');

```

```

<?
include('staffMenu.php');
include('TimelineLogin.php');

TimelineLogin::loadTimelineUsers();
$name=TimelineLogin::nameOfUser($userWanted);
echo"<h3>Staff: ".$name."</h3>";
?>
Records awaiting approval are listed below.<br>
<form method=post action='staffListArticles.php'>
<?
if ($approvedRecords=='YES')
    echo"<input type='checkbox' name='approvedRecords' value='YES' checked=true>";
else
    echo"<input type='checkbox' name='approvedRecords' value='YES'>";
?>
Also show approved records
<br><br><input type=submit value='reload'>
</form>
</body>
</html>

```

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

Run the website, log-in as staff and go to the staff page. Select the 'Approve or edit events' option. The staff member's name should be displayed, along with a checkbox and button.

The screenshot shows a web interface with two tabs: "View timeline" and "Set up new timeline". Below the tabs, the text "Staff: Dafydd Jones" is displayed. Underneath, it says "Records awaiting approval are listed below." followed by a checkbox labeled "Also show approved records" which is currently unchecked. At the bottom of the form is a "reload" button.

Before displaying the events, an additional method should be added to the TimelineCategory class. This will input the ID number for an event category and return the corresponding name, for example: category 4 of the History of Transport timeline will return the name 'Air'.

Open the **TimelineCategory.php** file and add the **nameOfCategory()** method shown below.

```

public static function nameOfCategory($categoryWanted,$categoryCount)
{
    $answer="";
    for ($i=1;$i<=$categoryCount;$i++)
    {
        if ($categoryWanted==TimelineCategory::$categories[$i]->symbol)
        {
            $answer = TimelineCategory::$categories[$i]->categoryName;
        }
    }
    return $answer;
}
?>

```

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

Return now to the **staffListArticles.php** file and add the lines of program code shown below. The program begins by setting up the column headings for a table to display event records. A loop checks all event objects and only displays those with a display code value of 1, indicating that they are awaiting staff approval. The contributor's website username is used to obtain their full forename and surname by means of the **nameOfUser()** method. The **nameOfCategory()** method obtains the category title from the symbol number.

```

Also show approved records
<br><br><input type=submit value='reload'>
</form>

<br><table border=0 cellpadding = 5>
<?
    echo"<tr><th class = outline>Timeline";
    echo"<th class = outline>Date";
    echo"<th class = outline>Category";
    echo"<th class = outline>Event";
    echo"<th class = outline>Submitted by";
    echo"<th class = outline>Submission date";
    echo"<th class = outline>Status";
    $eventCount = TimelineEvent::loadEvents();
    include('Timeline.php');
    $timelineCount=Timeline::loadTimelines();
    include('TimelineCategory.php');
    for ($i=1; $i<=$eventCount; $i++)
    {
        $status = TimelineEvent::$events[$i]->getDisplay();
        $display=false;
        if ($status ==1)
            $display=true;
        if($display==true)
        {
            $timelineID = TimelineEvent::$events[$i]->getTimelineID();
            $timelineTitle = Timeline::loadTitleByID($timelineID,$timelineCount);
            $categoryID = TimelineEvent::$events[$i]->getCategory();
            $contributorID = TimelineEvent::$events[$i]->getContributor();
            echo"<tr><td>".$timelineTitle;
            echo"<td>".TimelineEvent::$events[$i]->getStartYear();
            $categoryCount = TimelineCategory::loadByTimelineID($timelineID);
            $categoryName=TimelineCategory::nameOfCategory($categoryID,
                $categoryCount);
            echo"<td>".$categoryName;
            echo"<td width=400>".TimelineEvent::$events[$i]->getEventTitle();
            $contributor=TimelineLogin::nameOfUser($contributorID);
            echo"<td>".$contributor;
            echo"<td>".TimelineEvent::$events[$i]->getSubmissionDate();
            if ($status == 0)
                echo"<td>published";
            if ($status == 1)
                echo"<td>awaiting approval";
            echo"<td><input type=submit value='view'>";
        }
    }
    ?>
</table>

</body>
</html>

```


Save the **staffListArticles.php** file and copy it to the server. Refresh the staff page. A list of all event records awaiting approval should now be displayed.

Staff: Dafydd Jones

Records awaiting approval are listed below.
 Also show approved records

Timeline	Date	Category	Event	Submitted by	Submission date	Status	
History of Transport	1927	Air	Charles Lindbergh makes the first solo transatlantic flight	Angela Pritchard	29/02/2020	awaiting approval	<input type="button" value="view"/>
History of Transport	1959	Road	M1, the UK's first motorway	Angela Pritchard	29/02/2020	awaiting approval	<input type="button" value="view"/>

Return to the **staffListArticles.php** file and the add lines of code shown below. These allow the user to list all event records, including those which have already been approved.

```

for ($i=1; $i<=$eventCount; $i++)
{
    $status = TimelineEvent::$events[$i]->getDisplay();
    $display=false;
    if ($status ==1)
        $display=true;
    if (($approvedRecords=='YES') && ($status ==0))
        $display=true;
    if($display==true)
    {
        $timelineID = TimelineEvent::$events[$i]->getTimelineID();
    }
}

```

Save the **staffListArticles.php** file, copy it to the server and again refresh the staff page. Click the tick box to select 'Also show approved records', then click the 'Reload' button. All event records should now be listed.

Staff: Dafydd Jones

Records awaiting approval are listed below.
 Also show approved records

Timeline	Date	Category	Event	Submitted by	Submission date	Status	
History of Wales	1765	Industry	The heyday of Cyfarthfa ironworks, Merthyr Tydfil, under Robert Thompson Crawshay.	Angela Pritchard	04/06/2019	published	<input type="button" value="view"/>
History of Wales	1804	Transport	Richard Trevithick's steam locomotive runs from Penydarren Ironworks to the canal wharf at Abercynon.	Angela Pritchard	22/04/2019	published	<input type="button" value="view"/>
History of Wales	1808	Industry	Opening of the Hafod and Morfa copper works in the lower Swansea valley	Sam Mortlake	06/11/2019	published	<input type="button" value="view"/>
History of Wales	1837	Social history	Opening of Llanfyllin workhouse in response to the Poor Law Amendment Act of 1834.	Dafydd Jones	06/11/2019	published	<input type="button" value="view"/>

Each event record has been given a 'view' button which will open a page to display the record. To operate this, we must add a **<form>** structure to staffListArticles page.

Return to the **staffListArticles.php** file. Locate the start of the **if (\$display == true)** block and insert a line of code to open a form.

```

if (($approvedRecords=='YES') && ($status ==0))
    $display=true;
if($display==true)
{
    echo "<form method=post action = 'staffDisplayArticle.php'>";
    $timelineID = TimelineEvent::$events[$i]->getTimelineID();
}

```

Insert a closing block for the form at the end of the `if ($display == true)` block. Save `staffListArticles.php` and copy it to the server.

```

        if ($status == 1)
            echo"<td>awaiting approval";
            echo"<td><input type=submit value='view'>";
            $eventID=TimelineEvent::$events[$i]->getEventID();
            echo"<input type=hidden name = 'eventID' value=$eventID>";
            echo"</form>";
        }
    }
    ?>
</table>

```

It will also be necessary to add another method to the `TimelineEvent` class to update the display value of the record when an event is approved for display. Open `TimelineEvent.php` and add this method.

```

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

```

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


We can now move on to produce the new web page to display the event record. Open a blank file and add the lines of program code shown on the next page.

Save the file as `staffDisplayArticle.php` and copy it to the server. Run the staff webpage and select an event to display. The display page should open.

Contributor: Angela Pritchard
Submission date: 29/02/2020

Charles Lindbergh makes the first solo transatlantic flight

As Charles Lindbergh piloted the Spirit of St. Louis down the dirt runway of Roosevelt Field in New York on May 20, 1927, many doubted he would successfully cross the Atlantic Ocean. Yet Lindbergh landed safely in Paris less than 34 hours later, becoming the first pilot to solo a nonstop trans-Atlantic flight. He changed public opinion on the value of air travel, and laid the foundation for the future development of aviation. Lindbergh had persuaded nine St. Louis businessmen to finance his attempt, using their funds to build a special plane that Lindbergh helped design. Named in honor of his sponsors, the plane was called the Spirit of St. Louis.



Approved for display


```

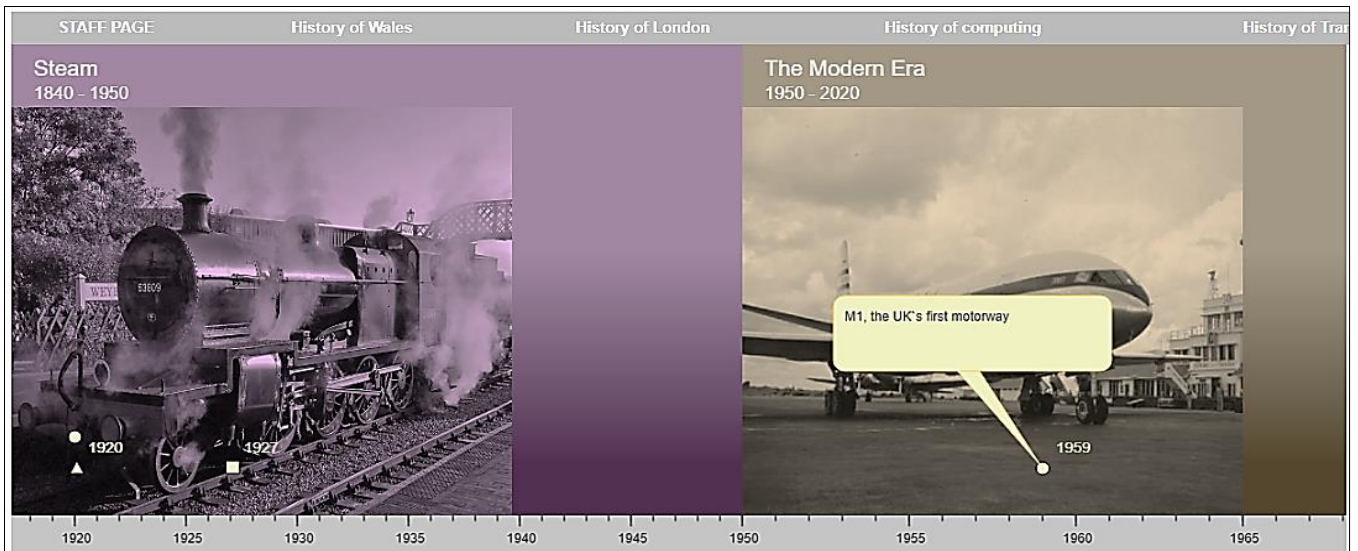
session_start();
include('TimelineEvent.php');
$userWanted=$_SESSION['user'];
$approvedRecords=$_REQUEST['approvedRecords'];

$edit=$_REQUEST['edit'];
if ($edit=='YES')
{
    $approved = $_REQUEST['approved'];
    $eventID = $_REQUEST['eventID'];
    $display=1;
    if ($approved == 'YES')
        $display=0;
    TimelineEvent::setDisplay($eventID,$display);
}
?>
<html>

```

Save the **staffListArticles.php** file and copy it to the server. Run the staff webpage and select an event to display. Use the tick box on the display page to change the approval status of the event record – either from 'not approved' to 'approved', or vice versa. Return to the staff page and check that the new status of the record is shown correctly in the table.

Go to the timeline and check that approved events are displayed, whilst events awaiting approval do not appear.



Further development

The web pages developed above provide the minimum functionality for a working historical timeline. A full system would need additional editing options, for example: adding or removing event categories, or allowing authors to edit the timeline event records.

This project demonstrates the design of a particular staff moderated on-line media system in which the published content is contributed by members. Many other group applications with material contributed by members could be developed. For example, web sites might document the activities of a sports club, or compile learning resources for students studying a particular course. Innovative graphical displays may provide access to the site content, such as a mind map illustrating a course syllabus with clickable links to articles on the various course topics.

Summary of the object structures

TimelineLogin

A TimelineLogin object contains the loginID which is set by the database as an auto-number, along with the person's name, email address, website user name and password, and a status variable identifying the user as member or staff. Two methods are included which check the input data for valid log-in details. The public method **checkPassword()** calls the private method **checkUser()** to examine each TimelineLogin object in turn, then returns a result of 'staff' or 'member' if valid log-in details are found for a user in one of these groups. An **addMember()** method allows new member records to be added to the database, and a **nameOfUser()** method allows a member's full name to be found from their web site user name.

Timeline

A Timeline object is created for each historical timeline which will be displayed on the website. Attributes specify the timeline title and the year range. Methods are provided to add timeline records to the database, and to retrieve records and create a corresponding set of Timeline objects. A series of **get()** methods allow access to the object attributes, such as timeline title, start and finish year.

TimelineCategory

Category objects represent the categories by which timeline events can be classified. The **addCategory()** method allows additional category records to be added to the database. A **loadByTimelineID()** method allows the set of TimelineCategory objects to be loaded for a particular timeline.

TimelinePeriod

A TimelinePeriod object is created for each of the time periods into which the timeline is divided. Attributes include the start and finish years for the period, the period name, and the horizontal year scale to be used when displaying the period on the timeline display. A file name specifies the background image to be used for the period. An **addPeriod()** method allows additional period records to be added to the database. A **loadByTimelineID()** method allows the set of TimelinePeriod objects to be loaded for a particular timeline. **Get()** methods the provide access to the object attributes.

TimelineEvent

A TimelineEvent object is created for each event record which will be displayed on the website. Attributes include the date(s) of the event, its title, a text description, and the file name of an image illustrating the event. The user name of the contributor is recorded, along with a **display** attribute which indicates whether the record has been approved for display on the web site. Methods are provided to add event records to the database, and to retrieve records and create a corresponding set of TimelineEvent objects for a particular timeline. A **setDisplay()** method allows the display status for a record to be changed when it is approved or rejected for display.

