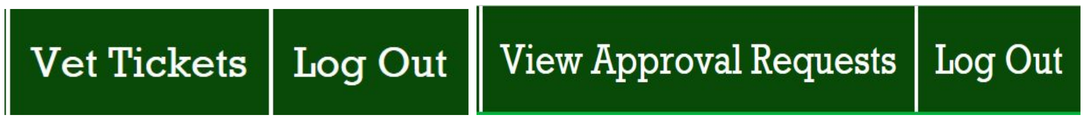
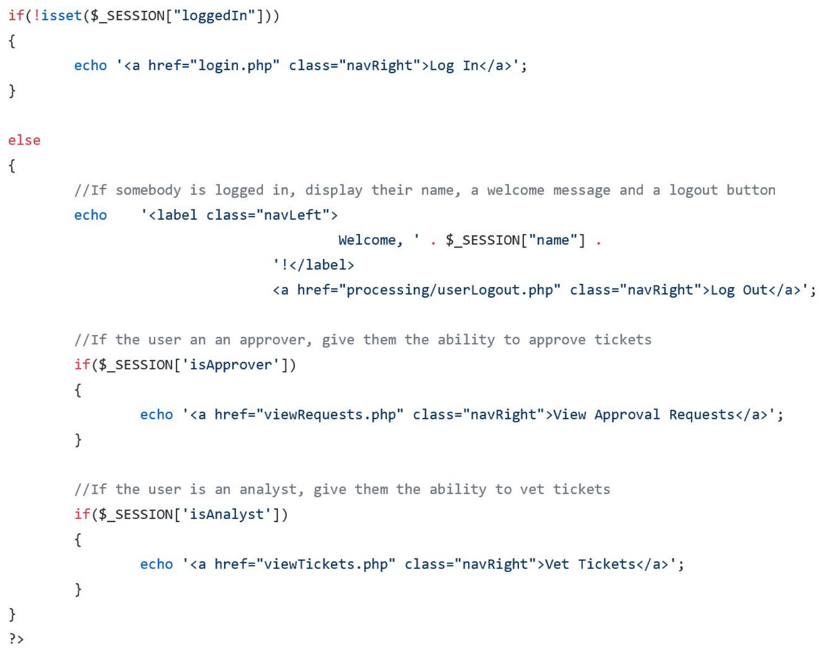
Team Covfefe’s review of Dancing to Zebras code

Taylen Jones, Scott Thomas, Jinpeng Chen

Overall Dancing To Zebras code was well written, and also very nicely commented, which made it an enjoyable experience to walk through and understand. They made great use of the design patterns and the website itself flowed nicely. It isn’t without flaws though, as there could be some cutdown on code duplication.

The team used a few design patterns, including: composite, state, decorator, observer and facade. The decorator pattern was well implemented in the navigation bar to display different options based on the permissions of the user that was logged in. The use of this pattern allowed the team so save time that would’ve been used typing out repetitions of already used code. Below you can see the view of an analyst (left) and approver (right). The code is seen below, simply using session variables to determine the correct info to display.

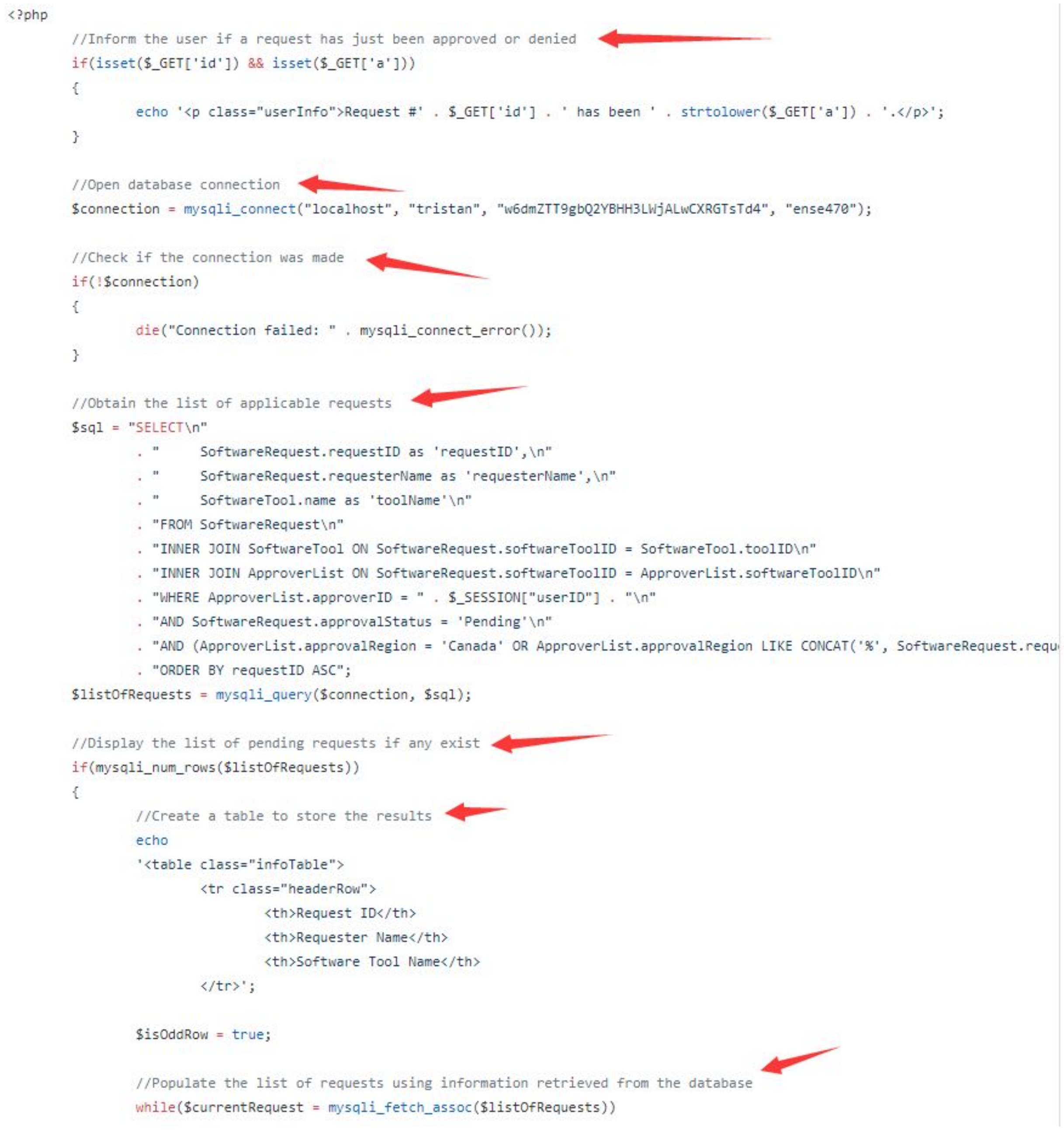




```
if(!isset($_SESSION["loggedIn"]))
{
        echo '<a href="login.php" class="navRight">Log In</a>';
}

else
{
        //If somebody is logged in, display their name, a welcome message and a logout button
        echo    '<label class="navLeft">
                                    Welcome, ' . $_SESSION["name"] .
                                '!</label>
                                <a href="processing/userLogout.php" class="navRight">Log Out</a>';

        //If the user an an approver, give them the ability to approve tickets
        if($_SESSION['isApprover'])
        {
                echo '<a href="viewRequests.php" class="navRight">View Approval Requests</a>';
        }

        //If the user is an analyst, give them the ability to vet tickets
        if($_SESSION['isAnalyst'])
        {
                echo '<a href="viewTickets.php" class="navRight">Vet Tickets</a>';
        }
}
?>
```

The team’s code is clean, all the codes have comments at the beginning of the sections. It is obvious for other programmers to understand.

Example in viewRequests.php

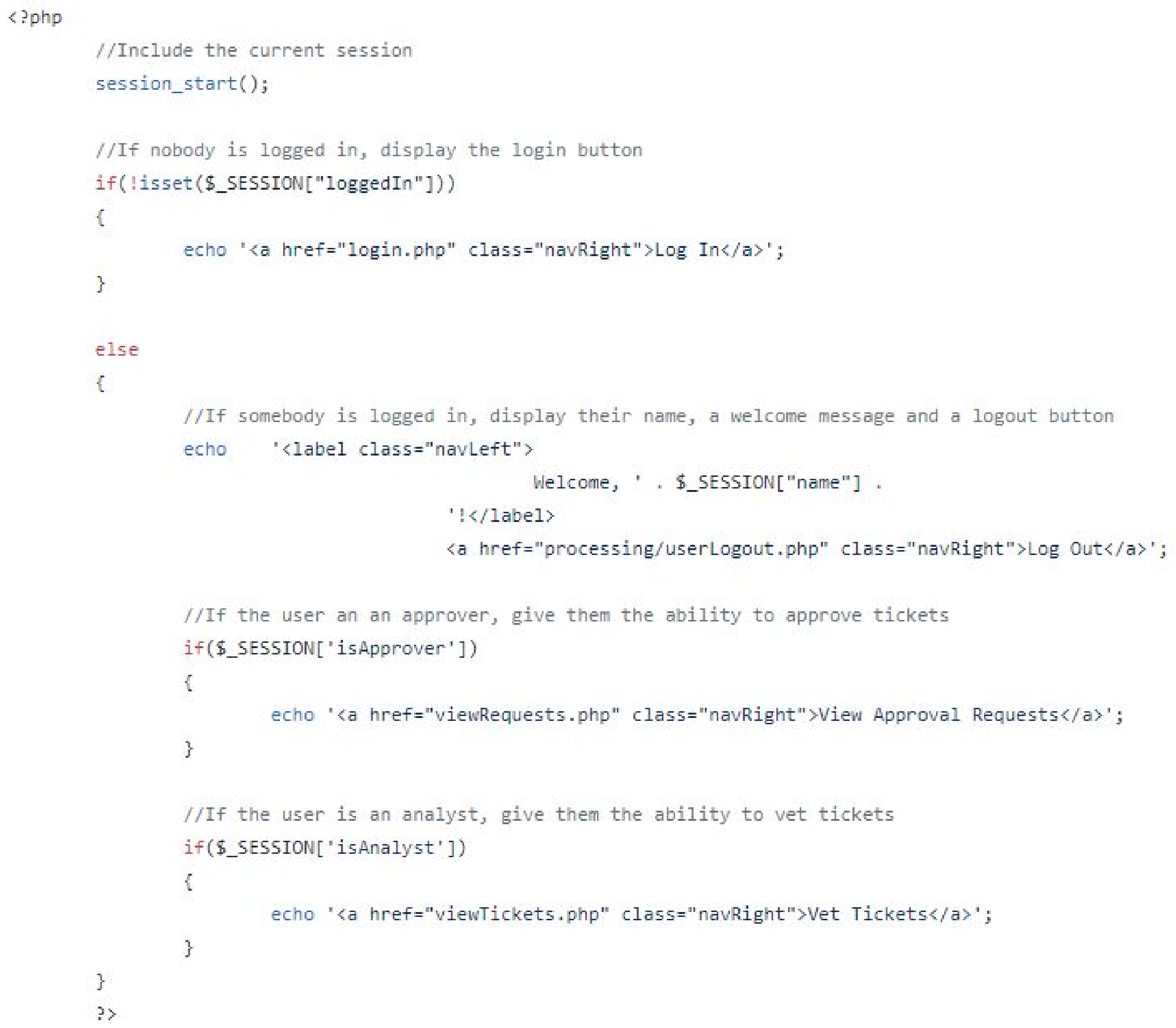


```php
<?php
	//Inform the user if a request has just been approved or denied
	if(isset($_GET['id']) && isset($_GET['a']))
	{
		echo '<p class="userInfo">Request #' . $_GET['id'] . ' has been ' . strtolower($_GET['a']) . '.</p>';
	}

	//Open database connection
	$connection = mysqli_connect("localhost", "tristan", "w6dmZTT9gbQ2YBHH3LWjALwCXRGTsTd4", "ense470");

	//Check if the connection was made
	if(!$connection)
	{
		die("Connection failed: " . mysqli_connect_error());
	}

	//Obtain the list of applicable requests
	$sql = "SELECT\n"
		. "	SoftwareRequest.requestID as 'requestID',\n"
		. "	SoftwareRequest.requesterName as 'requesterName',\n"
		. "	SoftwareTool.name as 'toolName'\n"
		. "FROM SoftwareRequest\n"
		. "INNER JOIN SoftwareTool ON SoftwareRequest.softwareToolID = SoftwareTool.toolID\n"
		. "INNER JOIN ApproverList ON SoftwareRequest.softwareToolID = ApproverList.softwareToolID\n"
		. "WHERE ApproverList.approverID = " . $_SESSION["userID"] . "\n"
		. "AND SoftwareRequest.approvalStatus = 'Pending'\n"
		. "AND (ApproverList.approvalRegion = 'Canada' OR ApproverList.approvalRegion LIKE CONCAT('%', SoftwareRequest.requ
		. "ORDER BY requestID ASC";
	$listOfRequests = mysqli_query($connection, $sql);

	//Display the list of pending requests if any exist
	if(mysqli_num_rows($listOfRequests))
	{
		//Create a table to store the results
		echo
		'<table class="infoTable">
			<tr class="headerRow">
				<th>Request ID</th>
				<th>Requester Name</th>
				<th>Software Tool Name</th>
			</tr>';

		$isOddRow = true;

		//Populate the list of requests using information retrieved from the database
		while($currentRequest = mysqli_fetch_assoc($listOfRequests))
```

Good variable naming, can easily understand what the variable is used for. Overall the code for each function is short and simple, which make the project easier to maintain.

However, there exists some code duplication. For example, the code for the session is seen in multiple php files. The easiest way to avoid this is have something like a header.php file and include/require this at the top of every page of the site that using session.

In the header.php file could have these code



```
<?php
        //Include the current session
        session_start();

        //If nobody is logged in, display the login button
        if(!isset($_SESSION["loggedIn"]))
        {
                echo '<a href="login.php" class="navRight">Log In</a>';
        }

        else
        {
                //If somebody is logged in, display their name, a welcome message and a logout button
                echo    '<label class="navLeft">
                                        Welcome, ' . $_SESSION["name"] .
                                '!</label>
                                <a href="processing/userLogout.php" class="navRight">Log Out</a>';

                //If the user an an approver, give them the ability to approve tickets
                if($_SESSION['isApprover'])
                {
                        echo '<a href="viewRequests.php" class="navRight">View Approval Requests</a>';
                }

                //If the user is an analyst, give them the ability to vet tickets
                if($_SESSION['isAnalyst'])
                {
                        echo '<a href="viewTickets.php" class="navRight">Vet Tickets</a>';
                }
        }
?>
```

Then when you need these codes, simply write <?php **require** "header.php"; ?>. This can help clean up the code.

As a whole, the program accomplishes all of the functionality deemed required in the minimum viable product. It also passes all ATDD cases and completes everything from the first release in the user story maps.

The state design pattern was effectively used to categorize requests based on their progress through the software distribution system. The composite pattern was used as an implementation of dynamic page generation for the software requests. Primarily, the usage of design patterns in the code was kept at a conceptual level, as opposed to a specific object-oriented programming example. As a result, the impact of the design patterns on the final end result was mostly trivial, but to no degradation of the program. Some of this disconnect is attributable to the usage of web programming languages and not java or c++ or similar.

Overall, the project avoids the majority of “code smells” and commits very few coding sins. Bloat is kept within reason, however “shotgun surgery” exists in part of the overall design. The aspects of code reuse within the program will all need to be altered separately for any change to be made. Specifically, the methods used to pull all ticket data from the database to dynamically generate pages all use very similar code. There are very few dispensibles in the code, as all the code seems to be used in the functionality of the program. Due to the design of the code and the usage of php there are no code smells in the category of “couplers”. As an entire program the code is very high quality, documented well, has no critical flaws, and accomplishes all of the initial goals from the outset.