http://ml-lab-4d78f073-aa49-4f0e-bce2-31e5254052c7.ukwest.cloudapp.azure.com:55803/index.php

This will bring you directly to the login page, as you have no cookie saying you are logged in, if you were it would bring you directly to the home page. The information as to how I constructed my sql databases is a direct child of the assessment folder called sql-database-creation-commands.txt

HTML pages for static content is there and validated (15 marks):

In the folders add_visit, home, login, overview, registration, report and settings the static HTML content is there. In each folder is a copy of the php file actually used to run the website called name(for-validation). This file contains only HTML and can be validated easier than sifting through the php files and separating the HTML from the php. I used php pages to actually use and display the content because you need php to access session variables and I wanted my file type to stay consistent.

CSS is used to style the pages according to the screenshots and style guide (15 marks):

Each page used to display content for the website is linked to 2 css documents. Styles.css which is in their respective folder and master.css which is in the css folder. The 2 files are used to create the layout that matches the screenshots.

The fonts are the same family and size as specified in the style guide.
The colours are the same rgb as specified in the style guide.
The watermark is similar to what it looks like in the picture with an opacity of 0.4.

PHP is correct (15):

To register works because you can register and then the user information is stored. This is done in the folder registration in the file registration-handler.php which calls functions from the folder php in the file functions.php. This then stores a users secured information in a sql database called users.

To check infection requires a lot of additional features I was unable to do, unfortunately without these it is hard to demonstrate my capability to pull them together with PHP.

To check overview in terms of PHP I understand how to do. I made a static HTML table with all the elemnts shown in the assessment brief. Unfortunately I did not figure out how to read a JSON file but if I did I would simply insert each value in where I currently have text, the PHP is not where I lack the understanding.

To login works because you can login after you have registered. This is done in the folder login in the file login-handler.php which calls functions from the folder php in the file functions.php. This allows access to the rest of the website.

To add a new visit works because you can add a new visit. This is done in the folder add-visit in the file add-visit-handler.php which calls function from the foler php in the file functions.php. The X and Y cords have default values that can be changed in the file add-visit-handler.php. This is because I did not know how to do the Jquery and Ajax call in order to make the image map submit form fields so I removed all of this from the code and just set default values. However the visit is added to an sql database called visits. I allowed multiple visits per user.

To report ann infection works because you can report and infection and the database infections is updated in response. This is done in the folder report in the file report-infection.php this calls functions from the folder php in the file functions.php. The infections are then stored in a database called infections. I allowed multiple infections per user, while this may be rare it is not unheard of.

To change the settings works because you can change the settings, these changes are stored in cookies named window and distance respectively. This is done in the folder settings in the file change-settings.php which calls functions from the folder php in the file functions.php.

Secure sessions and cookies (10 marks):

Session is created in the folder php in the file functions.php, (contains only php) in the function logInUser. It is created and the necessary session variables are assigned.

```
session_start();
   /*creating session variable*/
      $_SESSION['usersname'] = $uidExists["usersName"];
```

In order to be able to access the session on each page, at the top of each page there is the following php code.

```
<?php
  session_start();
?>
```

Session is destroyed after closing/logout. This is done in the folder php in the file logout.php(contains only php) with the following code. The code also ensures that the cookie that lets the user be seen as logged in is destroyed, as well as sending the user back to the login page. The cookies retaining the users settings are left untouched.

```
setcookie("is_logged_in", "true", time() -1 , "/");

session_start();
session_unset();
session_destroy();
header("location: ../login/login.php");
exit();
```

Cookies are used to store the settings. This is done firstly in the folder php in the file functions.php (contains only php) in the function createUser. The settings are given default values of 1 week and a distance of 100. This is done using the following code.

```
/*settings expiry set to 1 year*/
  setcookie("window", "1 week", time() + 60*60*24*365 , "/");
  setcookie("distance", 100 , time() + 60*60*24*365 , "/");
```

These settings are changed in the folder settings in the file change-settings.php.(contains only php) The user fills out a html form found in the file settings.php. This sends a post request to the file change-settings. The cookies are updated with the following code and the user is then sent back to settings.php

```
if (isset($_POST["window"])) {
    $window=$_POST["window"];
    setcookie("window", $window, time() + 60*60*24*365 , "/");
  }

  if (isset($_POST["distance"])) {
    $distance=$_POST["distance"];
    if ($distance >= 0 && $distance <= 500) {
      setcookie("distance", $distance , time() + 60*60*24*365 , "/");
    }
  }
  header("location: settings.php");
```

Security is Implemented (10 marks):

Salt is used in order to secure the password. This is done in the folder php in the file functions.php in the function createUser with the following code. Using the password_hash and then using the PASSWORD_BCRYPT as opposed to the PASSWORD_DEFAULT the password is first salted and then hashed. The cost is set to 10 in order to implement key stretching and makes the cryptography more secure against a brute force attack. The salted and then hashed password is stored into the database.

```
$options = [
    'cost' => 10,
  ];

  $hashedPassword = password_hash($password, PASSWORD_BCRYPT, $options);
```

The salt is used once again to compare the password in the database to what the user inputs when they login, this is done in the folder php in the file functions.php in the function logInUser. The salt is then extracted from the password from the database. This salt is added to the input from the user which is then hashed and compared to the value found in the database for the final password. This is done in the following code.

```
$passwordHashed = $uidExists["usersPwd"];

$checkpassword = password_verify($password, $passwordHashed);
```

The website is protected against cross site scripting in the folder php in the file functions.php in multiple functions. It is done using prepared statements. It is first done in the function uidExists, called from the file registration-handler.php in registration. This has the following code. It's purpose is to check if a username already exists but it also uses prepared statements and stores the data in an associative array that can be accessed later like in the code above. EX: $passwordHashed = $uidExists["usersPwd"];

```
$sql = "SELECT * FROM users WHERE usersUid = ?;";
  $stmt = mysqli_stmt_init($conn);

  if (!mysqli_stmt_prepare($stmt, $sql)) {
    ("location: registration.php?error=stmtfailed");
    exit();
  }

  mysqli_stmt_bind_param($stmt, "s", $username);
  mysqli_stmt_execute($stmt);

  $resultData = mysqli_stmt_get_result($stmt);

  if ($row = mysqli_fetch_assoc($resultData)) {
    return $row;
  } else {
    $result = false;
    return $result;
  }

  mysqli_stmt_close($stmt);
```

The use of prepared statements is also present in the function createUser

```
$sql = "INSERT INTO users (usersName, usersUid, usersPwd) VALUES (?, ?, ?);";
  $stmt = mysqli_stmt_init($conn);

  if (!mysqli_stmt_prepare($stmt, $sql)) {
    header("location: registration.php?error=stmtfailed");
    exit();
  }

  $options = [
    'cost' => 10,
  ];
```

```
$hashedPassword = password_hash($password, PASSWORD_BCRYPT, $options);

mysqli_stmt_bind_param($stmt, "sss", $name, $username, $hashedPassword);
mysqli_stmt_execute($stmt);
mysqli_stmt_close($stmt);
```

To protect against SQL injection I used the algorithm htmlentities() around every single text input that the user could give in, in order to escape the input or comments of the user and prevent malicious code from entering any database.

Javascript, Ajax and Web Service:
Unfortunately I was unable to implement these.

Comment for future assessments:

I personally found and from talking to others that the single area where most of the time was spent on was SQL. Nothing else came close, the assessment had a significant portion of it included. This is fine but we were explicitly told we weren't doing this and as a result not given the tools to be able to figure things out for ourself like with other features. I spent a lot of time and found that as a result of this I was unable to focus, learn and implement other features of the course that were highlighted and graded. I think it would be helpful for next year's students to either have this reduced, provided or taught in more detail.

This was especially true when it came to figuring out how you wanted it implemented with accessing it from the virtual machines, without the database and simply exactly what format we should hand it in as. This same confusion has been held by every person I have so far asked about it. I think it would be helpful to next year's students if this was more concrete.