Milestone 5

Team Members: Daniel Green, Darian Valdez, Zack Toelkes, Gayathri Gude, Gordon Gu, Joanne Liu

Project Name: Wanderlist

User Acceptance Test Plan

What we will test:

1.) **Create new user login**
   a.) Our application requires a new user to create a login to use our application. So a major feature in the finished product will to prove that we are able to actually create/save new users to the feature so they can login.
2.) **Final selection page**
   a.) A critical step in our application is when 5 locations have been “liked”, the user will be taken to a new page where these 5 “liked” locations will be displayed. Then the user is told to pick their top location. What will need to be tested for this step is that when 5 locations have been “liked”, the user will be taken to a new page to then choose their final location.
3.) **Correct data is pulled for each “liked” location**
   a.) The last major test that will be conducted will be showing that the correct data is pulled for each “liked” location. For example, if the user chose the restaurant locations filter, and they liked the restaurants: Applebees, Chilis, Smash Burger, Chipotle, and 5 guys, in that order, then the data stored in the counter will need to correctly pull the information for each of those restaurants respectively. So when the user is taken to the next page for their final selection, the information for each of their restaurants they “liked” will need to pop up in the final selection page.

What the test cases will be:

1.) **Create new user login**
   a.) How this will be tested is that we will determine if the create method executes without an error occuring. This will be done by manually entering a username that was recently created and determining if that user was saved within our database properly.
2.) **Final selection page**
   a.) This will be tested by updating a counter. When the user is “liking” locations, a counter will be updated every time a location is “liked”. When this counter reaches the value of 5, this will trigger the new page to pop up and then display all locations that have been chosen.

3.) Correct data is pulled for each “liked” location

a.) For the database that stores all 31 locations, we will create a 31 bit string and every bit in the bit string will correspond to a location in the database. So when a user “likes” a location, that bit in the bit string will be flipped to a “1”. This means that each “1” in the bit string represents a “liked” location. So then when the user has “liked” 5 locations and is taken to a new page, the software will scan the bit string and pull the information for each bit that has been flipped to a “1” and display it to the final selection page. This will ensure that each correct location has been pulled appropriately.

Automated Test Cases

1) User creation

This code will execute a create operation on the User datatable. If an error occurs, it is logged and the application is suspended. If it completes, the test is passed and the application continues

```
44
45          // GET: Users/Create
46          public IActionResult Create()
47          {
48              return View();
49          }
50
51          // POST: Users/Create
52          // To protect from overposting attacks, please enable the specific properties you want to bind to, for
53          // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
54          [HttpPost]
55          [ValidateAntiForgeryToken]
56          public async Task<IActionResult> Create([Bind("UserId,UserName,Password")] User user)
57          {
58              if (ModelState.IsValid)
59              {
60                  _context.Add(user);
61                  await _context.SaveChangesAsync();
62                  return RedirectToAction("Login", "", new { area = "" });
63              }
64              return View(user);
65          }
66
```

2) **Increment Locations**

This code will increment counts and update the users viewed bitstring. When a count of 5 locations is reached, the program will redirect to a “more info” page on the User Controller. Should this method fail, the application will be suspended and the error logged.

```
public ActionResult Next(int locId, int usrId, int count, int currentLoc, bool[] viewed)
{
    ViewData["Count"] = count + 1;
    viewed[locId] = true;

    BitArray b = new BitArray(viewed);
    int[] array = new int[1];
    b.CopyTo(array, 0);
    User usr = _context.User.Where(x => x.UserId == usrId).FirstOrDefault();
    usr.ViewedLocs = array[0];

    this.Edit(usrId, usr).RunSynchronously();

    if (count + 1 == 5)
    {
        //Redirect to more info
    }
    return View("Dashboard");
}
```

3) **Location Check**

This code checks to see if the location being sent to user is blank. If it is, an error is logged. If not, the location is allowed to pass through to whatever function is using it.

```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var location = await _context.Location
        .SingleOrDefaultAsync(m => m.LocationId == id);
    if (location == null)
    {
        return NotFound();
    }

    return View(location);
}
```