User Management

Estimated time needed: 1 hour and 20 mins

The admins of your app will need to manage users and user access based on users' types, such as visitors and registered users. Your task is to add a standard user management feature to your app.

Environment setup

If your lab workspace has been reset and you want to continue from where you left off previously, you can `git clone` or pull the latest code from your GitHub repository.

Set up the Python runtime again if the lab workspace has been reset. You can ignore this step if your lab environment is still active.

```
 1.  1
 2.  2
 3.  3
 4.  4
 5.  5
 6.  6
 7.  7
 8.  8
 9.  9
10.  10
11.  11
```

```
 1. cd /home/project/xrwvm-fullstack_developer_capstone/server
 2.
 3. pip install virtualenv
 4. virtualenv djangoenv
 5. source djangoenv/bin/activate
 6.
 7. python3 -m pip install -U -r requirements.txt
 8.
 9. python3 manage.py makemigrations
10.
11. python3 manage.py migrate
```

Copied!

Create a superuser for your app

Let's create a superuser first.

1. Stop the server if it is running.

2. Create a superuser by running the following command.

```
1.  1
```

```
1. python3 manage.py createsuperuser
```

Copied!

With `Username`, `Email`, and `Password` entered, you should see a message saying `Superuser created successfully.`, indicating that the superuser has been created.

3. Run the server by executing the following command.

```
1.  1
```

```
1. python3 manage.py runserver
```

Copied!

4. Click the `Django Admin` button below to open the admin panel.

Django Admin

   Note: If the button doesn't work, please launch the app on Port 8000 and append `/admin` at the end of the URL to get to the Django Admin UI.

5. Log in to the admin site with the credentials you just created for the superuser.

6. Click `Users` under the `AUTHENTICATION AND AUTHORIZATION` section. You should be able to view the superuser you just created.

Build the client-side and configure it

A starter code of the client has been provided for you.

1. Open a `New Terminal` and switch to the client directory.

   1. 1

   1. `cd /home/project/xrwvm-fullstack_developer_capstone/server/frontend`

   [Copied!]

2. Install all required packages.

   1. 1

   1. `npm install`

   [Copied!]

3. Run the following command to build the client.

   1. 1

   1. `npm run build`

   [Copied!]

4. Open `server/djangoproj/settings.py` in the editor. Under `TEMPLATES`, you will find `DIRS`. Add `os.path.join(BASE_DIR,'frontend/build'` to the list for the Django application to recognize the front end. It should be set up as below.

   1. 1
   2. 2
   3. 3
   4. 4
   5. 5

   ```
   1.        'DIRS': [
   2.            os.path.join(BASE_DIR, 'frontend/static'),
   3.            os.path.join(BASE_DIR, 'frontend/build'),
   4.            os.path.join(BASE_DIR, 'frontend/build/static'),
   5.        ],
   ```

   [Copied!]

5. In `server/djangoproj/settings.py`, add the directories for the Django application to look for static files to the list named `STATICFILES_DIRS`. It should be set up as below.

   1. 1
   2. 2
   3. 3
   4. 4
   5. 5

   ```
   1. STATICFILES_DIRS = [
   2.     os.path.join(BASE_DIR, 'frontend/static'),
   3.     os.path.join(BASE_DIR, 'frontend/build'),
   4.     os.path.join(BASE_DIR, 'frontend/build/static'),
   5. ]
   ```

   [Copied!]

Add a new login view

Next, you need to create a new login Django view to handle a login request.

1. Open `djangoapp/views.py` and uncomment the import statements in the top. Observe the login view to authenticate user. After the user logs in, it should return a JSON object with the username and status.

2. Open `server/djangoapp/urls.py` and uncomment the import statements in the top.

3. Configure a route for the login view by uncommenting the path entry in `server/djangoapp/urls.py`, as given below.

   1. 1

   1. `    path(route='login', view=views.login_user, name='login'),`

   [Copied!]

You may refer to this lab to get more details about Django authentication:

[Django Authentication System](#)

4. Configure a route for the login page by adding the path entry in `server/djangoproj/urls.py`, as given below. Login page is a REACT page rendered from a route that is configured in `/server/frontend/src/App.js`.

   1. 1

   1. `    path('login/', TemplateView.as_view(template_name="index.html")),`

   [Copied!]

4. Click the `Django App` button below to open the browser.

Django App

5. You will see the homepage. From the homepage click on the `Login` link provided. If you configured the Login page correctly, you should see the page as below.
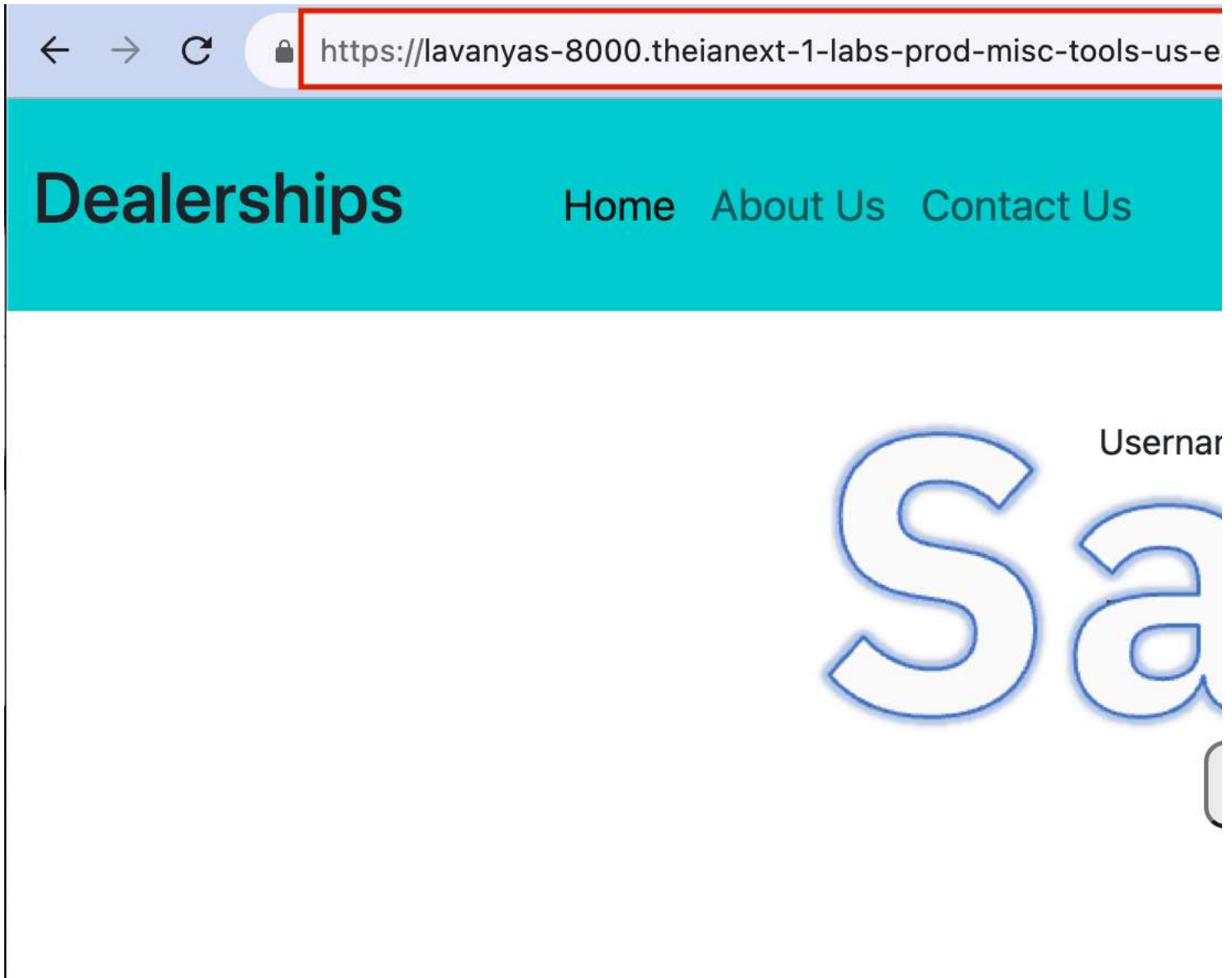
6. In the space provided, enter the username and password you gave for the superuser and login.

7. The login process should go through, and you should see username and logout option displayed on the homepage, as below.
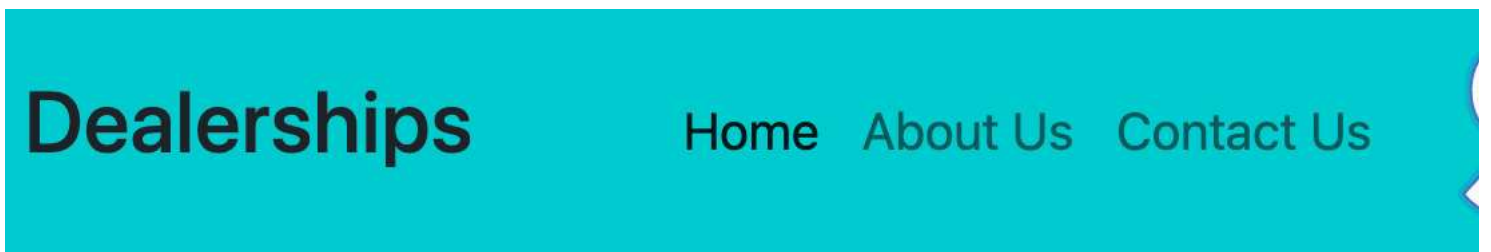
Take a screenshot for peer review

After you have created the login view, please take a screenshot and name it as `login.jpg` or `login.png` for peer review.

Add logout functionality

You need to create a new logout Django view to handle the logout request.

1. Open `djangoapp/views.py` and add a new logout view to handle a logout request. After the user logs out, it should return a JSON object with the username.

   1. 1

```
2.  2
3.  3
```

```
1.      logout(request)
2.      data = {"userName":""}
3.      return JsonResponse(data)
```

Copied!

2. Configure a route for the logout view by adding a path entry in `djangoapp/urls.py`.

3. In the `server/frontend/static/Home.html`, include the following code to handle the login out of the user, in the space provided.

```
1.  1
2.  2
3.  3
4.  4
5.  5
6.  6
7.  7
8.  8
9.  9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
```

```
1.   let logout_url = window.location.origin+"/djangoapp/logout";
2.   const res = await fetch(logout_url, {
3.     method: "GET",
4.   });
5.
6.   const json = await res.json();
7.   if (json) {
8.     let username = sessionStorage.getItem('username');
9.     sessionStorage.removeItem('username');
10.    window.location.href = window.location.origin;
11.    window.location.reload();
12.    alert("Logging out "+username+"...")
13.   }
14.   else {
15.    alert("The user could not be logged out.")
16.   }
```

Copied!

4. Open a new terminal and run the following command.

```
1.  1
2.  2
```

```
1. cd /home/project/xrwvm-fullstack_developer_capstone/server/frontend
2. npm run build
```

Copied! Executed!

5. Click `Django Application` below to open the browser.

Django Application

6. If the session didn't expire, you will be logged in from the previous step. If not, login, and then, click `Logout`.

You should see an alert showing that you have successfully logged out.

Note: In case the Logout alert doesn't appear, try checking its functionality by switching to a different browser.
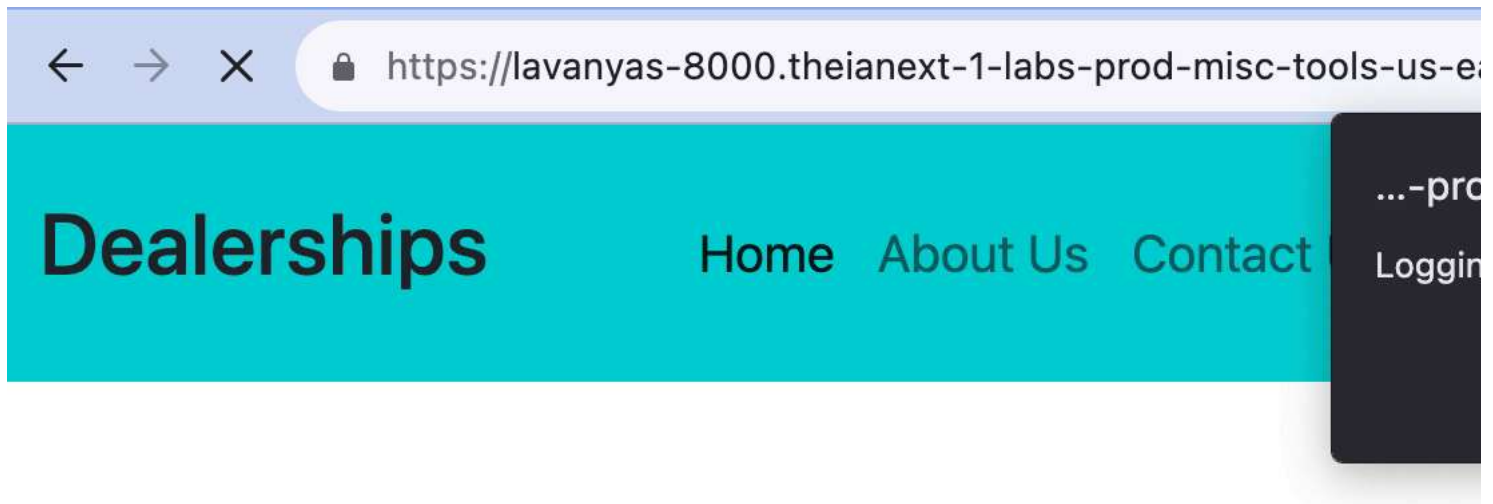
Take a screenshot for peer review

After you have created the logout view, please take a screenshot and name it `logout.jpg` or `logout.png` for peer review.

At this point, you should be able to log in and log out with the superuser you created earlier.

Add a 'Register' functionality

You need to create a new registration Django view to handle the register request.

1. Open `djangoapp/views.py` and add a new register view to handle a register request. When the user registers, a user object should be created, and the user should be logged in. It should return a JSON object with the username.

▼ Click here for a suggested implementation

```
1.  1
2.  2
3.  3
4.  4
5.  5
6.  6
7.  7
8.  8
9.  9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
```

```python
1.  @csrf_exempt
2.  def registration(request):
3.      context = {}
4.
5.      data = json.loads(request.body)
6.      username = data['userName']
7.      password = data['password']
8.      first_name = data['firstName']
9.      last_name = data['lastName']
10.     email = data['email']
11.     username_exist = False
12.     email_exist = False
13.     try:
14.         # Check if user already exists
15.         User.objects.get(username=username)
16.         username_exist = True
17.     except:
18.         # If not, simply log this is a new user
19.         logger.debug("{} is new user".format(username))
20.
21.     # If it is a new user
22.     if not username_exist:
23.         # Create user in auth_user table
24.         user = User.objects.create_user(username=username, first_name=first_name, last_name=last_name,password=password, email=email)
25.         # Login the user and redirect to list page
26.         login(request, user)
27.         data = {"userName":username,"status":"Authenticated"}
28.         return JsonResponse(data)
29.     else :
30.         data = {"userName":username,"error":"Already Registered"}
31.         return JsonResponse(data)
```

[Copied!]

2. Configure a route for the registration view by adding a path entry in `djangoapp/urls.py`.

3. Inside `frontend/src/components/Register`, create a file named `Register.jsx`. The CSS to be used for this page has already been provided.

4. Add the following code in Register.jsx.
   Feel free to make changes to the look and feel.

```
1.  1
2.  2
3.  3
4.  4
```

       5. 5
       6. 6
       7. 7
       8. 8
       9. 9
      10. 10
      11. 11
      12. 12
      13. 13
      14. 14
      15. 15
      16. 16
      17. 17
      18. 18
      19. 19
      20. 20
      21. 21
      22. 22
      23. 23
      24. 24
      25. 25
      26. 26
      27. 27
      28. 28
      29. 29
      30. 30
      31. 31
      32. 32
      33. 33
      34. 34
      35. 35
      36. 36
      37. 37
      38. 38
      39. 39
      40. 40
      41. 41
      42. 42
      43. 43
      44. 44
      45. 45
      46. 46
      47. 47
      48. 48
      49. 49
      50. 50
      51. 51
      52. 52
      53. 53
      54. 54
      55. 55
      56. 56
      57. 57
      58. 58
      59. 59
      60. 60
      61. 61
      62. 62
      63. 63
      64. 64
      65. 65
      66. 66
      67. 67
      68. 68
      69. 69
      70. 70
      71. 71
      72. 72
      73. 73
      74. 74
      75. 75
      76. 76
      77. 77
      78. 78
      79. 79
      80. 80
      81. 81
      82. 82
      83. 83
      84. 84
      85. 85
      86. 86
      87. 87
      88. 88
      89. 89
      90. 90
      91. 91
      92. 92
      93. 93
      94. 94
      95. 95
      96. 96
      97. 97
      98. 98

       1. import React, { useState } from "react";
       2. import "./Register.css";
       3. import user_icon from "../assets/person.png"

```
 4.  import email_icon from "../assets/email.png"
 5.  import password_icon from "../assets/password.png"
 6.  import close_icon from "../assets/close.png"
 7.
 8.  const Register = () => {
 9.
10.     const [userName, setUserName] = useState("");
11.     const [password, setPassword] = useState("");
12.     const [email, setEmail] = useState("");
13.     const [firstName, setFirstName] = useState("");
14.     const [lastName, setlastName] = useState("");
15.
16.
17.     const gohome = ()=> {
18.         window.location.href = window.location.origin;
19.     }
20.
21.     const register = async (e) => {
22.         e.preventDefault();
23.
24.         let register_url = window.location.origin+"/djangoapp/register";
25.
26.         const res = await fetch(register_url, {
27.             method: "POST",
28.             headers: {
29.                 "Content-Type": "application/json",
30.             },
31.             body: JSON.stringify({
32.                 "userName": userName,
33.                 "password": password,
34.                 "firstName":firstName,
35.                 "lastName":lastName,
36.                 "email":email
37.             }),
38.         });
39.
40.         const json = await res.json();
41.         if (json.status) {
42.             sessionStorage.setItem('username', json.userName);
43.             window.location.href = window.location.origin;
44.         }
45.         else if (json.error === "Already Registered") {
46.             alert("The user with same username is already registered");
47.             window.location.href = window.location.origin;
48.         }
49.  };
50.
51.     return(
52.         <div className="register_container" style={{width: "50%"}}>
53.             <div className="header" style={{display: "flex",flexDirection: "row", justifyContent: "space-between"}}>
54.                 <span className="text" style={{flexGrow:"1"}}>SignUp</span>
55.                 <div style={{display: "flex",flexDirection: "row", justifySelf: "end", alignSelf: "start" }}>
56.                 <a href="/" onClick={()=>{gohome()}} style={{justifyContent: "space-between", alignItems:"flex-end"}}>
57.                     <img style={{width:"1cm"}} src={close_icon} alt="X"/>
58.                 </a>
59.                 </div>
60.                 <hr/>
61.             </div>
62.
63.             <form onSubmit={register}>
64.             <div className="inputs">
65.                 <div className="input">
66.                     <img src={user_icon} className="img_icon" alt='Username'/>
67.                     <input type="text"  name="username" placeholder="Username" className="input_field" onChange={(e) => setUserName(e.target.value)}/>
68.                 </div>
69.                 <div>
70.                     <img src={user_icon} className="img_icon" alt='First Name'/>
71.                     <input type="text"  name="first_name" placeholder="First Name" className="input_field" onChange={(e) => setFirstName(e.target.value)}/>
72.                 </div>
73.
74.                 <div>
75.                     <img src={user_icon} className="img_icon" alt='Last Name'/>
76.                     <input type="text"  name="last_name" placeholder="Last Name" className="input_field" onChange={(e) => setlastName(e.target.value)}/>
77.                 </div>
78.
79.                 <div>
80.                     <img src={email_icon} className="img_icon" alt='Email'/>
81.                     <input type="email"  name="email" placeholder="email" className="input_field" onChange={(e) => setEmail(e.target.value)}/>
82.                 </div>
83.
84.                 <div className="input">
85.                     <img src={password_icon} className="img_icon" alt='password'/>
86.                     <input name="psw" type="password"  placeholder="Password" className="input_field" onChange={(e) => setPassword(e.target.value)}/>
87.                 </div>
88.
89.             </div>
90.             <div className="submit_panel">
91.                 <input className="submit" type="submit" value="Register"/>
92.             </div>
93.             </form>
94.         </div>
95.     )
96.  }
97.
98.  export default Register;
```

[Copied!]

5. Configure the routes in `frontend/src/App.js`.

6. In the terminal you used previously to run `npm run build`, run it again for the latest changes to be reflected.

7. Open `server/djangoproj/urls.py` in the editor and add the following path to the `urlpatterns`. The routes have already been configured in `App.js`.
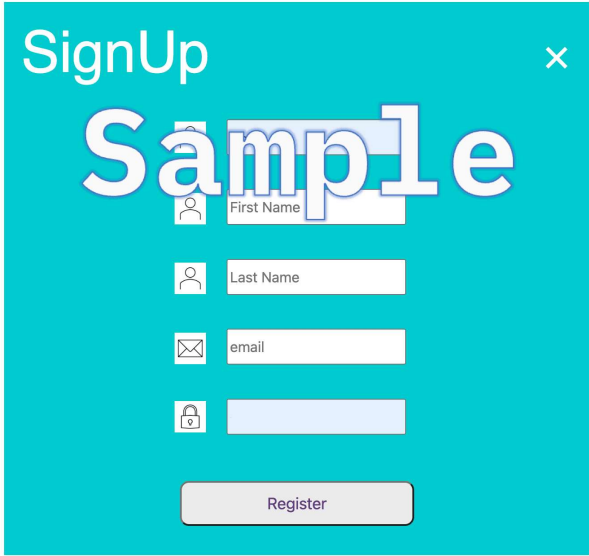
1. 1

1.   `path('register/', TemplateView.as_view(template_name="index.html")),`

Copied!

8. Click the `Django Application` button below to open the browser.

Django Application

Logout if you are logged in and click on the `Register` link. You should see a sign-up page as shown below.

9. Take a screenshot for peer review

After you have created the sign-up view, please take a screenshot and name it `sign-up.jpg` or `sign-up.png` for peer review.

10. Enter user details and register. It should sign up, login, and return an active session.

Test and Submit

1. Now you may want to test the updated Django app with signup, login, and logout end-to-end.

2. Commit and push the changes to your GitHub repository.

If you need to refresh your memory on how to commit and push to GitHub, please refer to this lab Working with git

3. In this lab, you have added user management-related templates and views to the app. In the next lab, you will start to create car/make related models, templates, and views.

Author(s)

Lavanya

Other Contributor(s)

Upkar Lidder