April 30, 2023

Rasool abbas, Onel Jimenez, Michael Mcfarlin, Ethan Wharton

Project rome

Test Scenario Specification

Test documentation

Fundraiser editing:

|  |  |
| --- | --- |
| **Title** | EditFundraiser\_FAIL\_NoUserClaim() |
| **Feature/Subfeature** | Edit fundraiser |
| **Purpose** | To test if the system throws an exception when a user is not logged in and tries to edit a fundraiser. |
| **Initial Conditions** | User is not logged in |
| **Test Data** | invalidFundraiserID: a randomly generated invalid fundraiser ID.  mockEditFundraiserMessage: a mock EditFundraiserMessage with random data. |
| **Test Actions** | Try to run EditFundraiser() with above arguments |
| **Expected Results** | Should throw an exception indicating no user claim. |

|  |  |
| --- | --- |
| **Title** | EditFundraiser\_FAIL\_InvalidUserClaim() |
| **Feature/Subfeature** | Edit fundraiser |
| **Purpose** | To test if the system throws an exception when attempting to edit a fundraiser with an invalid user claim. |
| **Initial Conditions** | App is running |
| **Test Data** | Mock UserID, FundraiserID, and EditFundraiserMessage are created as well as a \_mockHttpContext list. |
| **Test Actions** | Try to run EditFundraiser() with above arguments |
| **Expected Results** | Should throw an exception indicating invalid user claim. |

|  |  |
| --- | --- |
| **Title** | EditFundraiser\_FAIL\_UserNotFoundByClaim() |
| **Feature/Subfeature** | Edit fundraiser |
| **Purpose** | To test if the system throws an exception when attempting to edit a fundraiser with an invalid user claim. |
| **Initial Conditions** | App is running |
| **Test Data** | Mock UserID, FundraiserID, and EditFundraiserMessage are created as well as a \_mockHttpContext list. |
| **Test Actions** | Try to run EditFundraiser() with above arguments |
| **Expected Results** | Should throw an exception indicating user not found by user claim. |

|  |  |
| --- | --- |
| **Title** | EditFundraiser\_FAIL\_FundraiserNotFound() |
| **Feature/Subfeature** | Edit fundraiser |
| **Purpose** | To test that an exception is thrown when attempting to edit a fundraiser that does not exist in the system |
| **Initial Conditions** | App is running |
| **Test Data** | Mock User, invalidFundraiserID, and EditFundraiserMessage are created as well as a \_mockHttpContext list. |
| **Test Actions** | Try to run EditFundraiser() with above arguments |
| **Expected Results** | Should throw an exception indicating fundraiser not found. |

|  |  |
| --- | --- |
| **Title** | EditFundraiser\_FAIL\_UserNotAuthorizedToEditFundraiser() |
| **Feature/Subfeature** | Edit fundraiser |
| **Purpose** | To test whether an exception is thrown when a user attempts to edit a fundraiser that they are not authorized to edit. |
| **Initial Conditions** | App is running |
| **Test Data** | Valid EditFundraiser method is created, valid User object and Fundraiser with different values as user. |
| **Test Actions** | Try to run EditFundraiser() with above arguments |
| **Expected Results** | Should throw an exception indicating invalid credentials/unauthorized. |

|  |  |
| --- | --- |
| **Title** | EditFundraiser\_Success() |
| **Feature/Subfeature** | Edit fundraiser |
| **Purpose** | To ensure that with proper input a fundraiser can be created successfully. |
| **Initial Conditions** | App is running |
| **Test Data** | Valid User, Fundraiser, and EditFundrasierMessage is created |
| **Test Actions** | Try to run EditFundraiser() with above arguments |
| **Expected Results** | No exceptions should be thrown |

Fundraiser creation:

|  |  |
| --- | --- |
| **Title** | CreateFundraiser\_FAIL\_NoUserClaim() |
| **Feature/Subfeature** | Fundraiser Creation |
| **Purpose** | To test whether an exception is thrown if the user claim is missing while trying to create a fundraiser. |
| **Initial Conditions** | App is running |
| **Test Data** | A CreateFundraiserMessage object with randomly generated data.  An empty list of claims to simulate the absence of user claim. |
| **Test Actions** | Assert that the correct exception is thrown |
| **Expected Results** | An exception will be thrown indicating that no user claim was found. |

|  |  |
| --- | --- |
| **Title** | CreateFundraiser\_FAIL\_InvalidUserClaim() |
| **Feature/Subfeature** | Fundraiser Creation |
| **Purpose** | To test whether an exception is thrown if the user claim is invalid while trying to create a fundraiser. |
| **Initial Conditions** | App is running |
| **Test Data** | A mock CreateFundraiserMessage object with randomly generated values for Title, Description, Goal, ExpirationDate, and Category.  A randomly generated invalid user ID.  A list of claims for the mock HTTP context, containing a single claim with the invalid user ID.  A null user returned by the mock authentication business logic when GetUserByID is called with the invalid user ID. |
| **Test Actions** | Assert that the correct exception is thrown |
| **Expected Results** | An exception will be thrown indicating that no valid user claim was found. |

|  |  |
| --- | --- |
| **Title** | CreateFundraiser\_Success() |
| **Feature/Subfeature** | Fundraiser Creation |
| **Purpose** |  |
| **Initial Conditions** | App is running |
| **Test Data** | Mock User and Fundraiser message data are created.  Mock fundraiser ID is created. |
| **Test Actions** | Call the CreateFundraiser method of the FundraiserBusinessLogic with the mock create fundraiser message. |
| **Expected Results** | Fundraiser ID in the response message is equal to the mock fundraiser ID |

|  |  |
| --- | --- |
| **Title** | create-fundraiser-spec.ts |
| **Feature/Subfeature** | Fundraiser Creation |
| **Purpose** | End-to-end test emulating how a user might create an arbitrary fundraiser. Steps through each step of the process and ensures that that each task is completed successfully. |
| **Initial Conditions** | Site/API/DB must be up and running, must be able to connect to frontend. |
| **Test Data** | Generate random fundraiser details (title, description, category, end date, and goal) using faker library. |
| **Test Actions** | 1. Navigate to the login page of the web app. 2. Enter the user's email and password to log in. 3. Click on the "Create fundraiser" button. 4. Fill in the "Fundraiser name" input field with the previously generated title. 5. Click on the "Fundraiser goal" input field and fill in the previously generated goal amount. 6. Fill in the "End date" input field with the previously generated end date. 7. Select the previously generated category from the dropdown menu. 8. Fill in the "Description of fundraiser" input field with the previously generated description. 9. Click on the "Create fundraiser" button to create the fundraiser. 10. Verify that the newly created fundraiser is displayed correctly on the page, including its title, description, end date, and goal amount. |
| **Expected Results** | New fundraiser with random attributes should be created. |

User Login API:

|  |  |
| --- | --- |
| **Title** | LoginUser\_FAIL\_UserNotFound() |
| **Feature/Subfeature** | User Login |
| **Purpose** | Ensure correct behavior when a user account is not found. |
| **Initial Conditions** | An empty user database |
| **Test Data** | A LoginUserMessage object with fake email and password |
| **Test Actions** | Sets up the user data access object to return null when GetUserByEmail is called |
| **Expected Results** | Assert.Throws<HttpResponseException> when \_authBusinessLogic.LoginUser is called |

|  |  |
| --- | --- |
| **Title** | LoginUser\_FAIL\_WrongPassword() |
| **Feature/Subfeature** | User Login |
| **Purpose** | Ensure correct behavior when a user password is not incorrect. |
| **Initial Conditions** | A user with a hashed password |
| **Test Data** | A LoginUserMessage object with a fake email and an incorrect password |
| **Test Actions** | Sets up the user data access object to return the fake user when GetUserByEmail is called |
| **Expected Results** | Assert.Throws<HttpResponseException> when \_authBusinessLogic.LoginUser is called |

|  |  |
| --- | --- |
| **Title** | LoginUser\_Success() |
| **Feature/Subfeature** | User Login |
| **Purpose** | Ensure correct login behavior when user is found and password correct. |
| **Initial Conditions** | A user with a hashed password |
| **Test Data** | A LoginUserMessage object with the user's email and password |
| **Test Actions** | Sets up the user and authentication data access objects to return the fake user and user role |
| **Expected Results** | Assert.True when \_authBusinessLogic.LoginUser is called, verifying that the returned authToken is not null or empty |

User Registration:

|  |  |
| --- | --- |
| **Title** | register.spec.ts |
| **Feature/Subfeature** | User Registration |
| **Purpose** | To test the user registration process. |
| **Initial Conditions** | The application is running and reachable at http://localhost:3000/ |
| **Test Data** | Mock first name, last name, email, and password are generated |
| **Test Actions** | 1. Navigate to http://localhost:3000/. 2. Click on the Log on button button. 3. Click on the Create one here! link. 4. Fill the First Name Input Field with mock first name. 5. Press Tab to move to the Last Name Input Field. 6. Fill the Last Name Input Field with mock last name. 7. Press Tab to move to the Email Input Field. 8. Fill the Email Input Field with the mock email. 9. Press Tab to move to the Password Input Field. 10. Fill the Password Input Field with the mock password. 11. Press Tab to move to the Confirm Password Input Field. 12. Fill the Confirm Password Input Field with the mock password. 13. Click on the Register Button. 14. Click on the Email Input Field. 15. Fill the Email Input Field with the mock email. 16. Press Tab to move to the Password Input Field. 17. Fill the Password Input Field with the mock password. 18. Click on the Login Button. 19. Click on the Welcome, {mockFirstName} {mockLastName} text. |
| **Expected Results** | Expected Result: The user should be able to register and login successfully, and the Welcome, {mockFirstName} {mockLastName} text should be present on the page. |