Testing Strategy - OCS Mobile App

Test Approach

Tools - Manual Testing/Validation

BrowserStack will be the tool of choice when initially manually testing and validating the front-end solution. BrowserStack will provide relatively instant feedback from the latest real-world devices and operating systems.

Tools - Automation Testing

For testing the react based front-end SPA the automation tools that will be used will be based on CodeceptJS. CodeceptJS allows for react specific UI locators whereby elements can be identified by name, props and state. CodeceptJS also allows mobile testing using Appium which can also be configured to use BrowserStack if necessary. There are also multiple output types which can be configured to feed into CI/CD pipelines and display an output of test results. Another advantage of this framework is use of multiple helpers within one solution, e.g. Use of the REST helper and Appium helper is possible. In the instance of this OCS app, the QA will be able to make a REST request to fetch specific data, then within Appium it can be validated that the data on screen matches that of the returned request.

Role

The QA Engineer will be involved during the initial spec/story inception part of the development lifecycle. It is important for the QA to be clear on what the acceptance criteria of each user story is whilst at the same the QA can often think of the possible sad-path and edge-case scenarios that have yet to be defined. If the Agile process is being followed the QA will also be required to be involved during spec/story refinement sessions. During these sessions with the wider team the QA will be able to identify the parts of the acceptance criteria where the E2E tests will be involved and if test automation can be applied, if so the QA can offer advice to developers where specific interactable front-end elements may need prior-defined "*data-ids*" on, e.g for a login button:

```
<button type="submit" class="button" data-id="LoginButton">Login</button>
```

Following this method will allow the QA to define any locators, test methods/functions simultaneously at the same time development is going on with the intention of having a 'ready to run' test by the time a user story gets promoted to the test environment.

If a spec has already been provided and approved before this then the role of QA is to identify the necessary acceptance criteria and author the appropriate automation tests based on that which cover the happy-path, and any sad-path and edge case scenarios. If there are any cases whereby automation is not possible then manual test scripts should be documented to cover the gaps.

Defect Management

If any defects or bugs are identified that do not conform with the specification then these will be logged and discussed with the development team and moved back into the sprint as priority. It's important for the development team and QA to identify possible regression scenarios from this and define specific test cases to cover these instances. If an issue is identified in sprint but has not been considered as part of the specification the issue will be discussed with the development team and potentially turned into a backlog item to work on for a future sprint.

Release Management

* Based on a environment flow of, Local > Development > Stage > Production

Once any user story or front-end code has been developed and released on to the development environment the QA can begin either validating the prior-authored automation tests or begin writing the automation tests. In order to promote from the existing environment to the next there must be a 100% test pass rate with any previous bugs fixed (if any).

Ideally the CI/CD pipelines will be hooked up to trigger the automation tests on the development environment any time there is a release from local to development; these automation tests can be pre-defined smoke tests which will quickly identify either code or deployment issues without requiring to run a full test suite, if those tests then pass a full regression suite can be run to ensure full confidence. This process should also be followed development to stage.

Additional Comments/Remarks

- API does not handle exception scenarios with regards to 404's or 400 response codes. For example if the application is web-based and the URL can be manipulated by the user, a possibility is that they can enter non-existing, or invalid formatted athlete_id's or game_id's; this should be handled to appropriately display an error/exception through the front end.
- athlete_id is defined as a string, but game_id defined as a number. Should all "ids" be defined by numbers? Meant athletes returned not in id order.
- Spec suggests pagination in the front-end but the current API does not support this
    - Normally this is defined within the request parameters
- Part of the API Spec specified "Fourths" to be returned by this is not referenced by the database schema

Example Test Cases

Validate athletes display on opening of app

- Given I open the OSC app
- When the app fully loads
- I can page titled ‘Olympic Athletes’

Validate games ordered recent to oldest

- Given I open the OSC app
- When the app fully loads
- I can see a scrollable list of games
    - Ordered most recent games to oldest games
    - Event ‘name’ and ‘year’ content displayed

Validate athletes display in scrollable horizontal carousel

- Given I open the OSC app
- When the app fully loads
- I can see a scrollable horizontal carousel of athletes
    - Grouped by games
    - Ordered by global score (will need calculating prior)
    - Carousel is horizontally scrollable
    - Athlete ‘cards’ display athlete thumbnail photo
    - Athlete ‘cards’ are clickable

Validate athlete details page

- Given I open the OSC app
- When I click on an Athlete card
- I can see page titled ‘{athlete_name} details’
    - Photo displayed left
    - Name, DoB, Weight (Kg), Height (cm) displayed right
    - Medals section displayed
        - Ordered by Games, recent first
        - Displaying Gold, Silver, Bronze medal count per game
    - Bio displayed