You recently applied for a role of Automation Test Engineer and have made it to the short list of candidates. Since we are looking for someone who can create an automated test framework, we require you to complete the following exercise. The task consists of a programming exercise and some related questions.

**Scenario**

Our developers have finished coding a borrowing calculator and would like to make sure that the calculator continues to work as they make other changes to the page. They have asked you to build some automated tests covering various use cases to verify that the calculator is working properly. They intend to run these tests as part of every build.

They have asked you to use Cucumber, with UI test automation tool of your choice (e.g. Puppeteer, WebdriverIO, Selenium Webdriver, etc) and language of your choice selected between JavaScript or Java. The tests should run in a browser of your choice.

The current working production page is here: <https://www.anz.com.au/personal/home-loans/calculators-tools/much-borrow/>

**Exercise**

The exercise is to create a Java/JavaScript project (or projects) in Eclipse/Intellij, add Cucumber and any other add-ins you might need.

Develop the following three tests:

1. A person with the following details:
   1. Single,
   2. 0 dependants,
   3. buying a home to live in,
   4. with income of $80,000,
   5. other income $10,000,
   6. living expenses $500,
   7. current home loan repayments $0,
   8. other loan repayments $100,
   9. other commitments $0
   10. and total credit card limits $10,000

has a borrowing estimate of $479,000.

1. Clicking the ‘start over’ button clears the form.
2. Entering only $1 for Living expenses, and leaving all other fields as zero, clicking ‘Work out how much I could borrow’ returns the following message:  
   “Based on the details you've entered, we're unable to give you an estimate of your borrowing power with this calculator. For questions, call us on 1800 100 641.”

You will need to write JavaScript/Java code to interact with your selected UI test automation tool and to create commands within Cucumber using Behaviour-Driven Development (BDD).

All tests should run and pass. Please complete the solution and share it via gitHub with instructions on how to run the project. The repository should be shared with the recruiting manager.

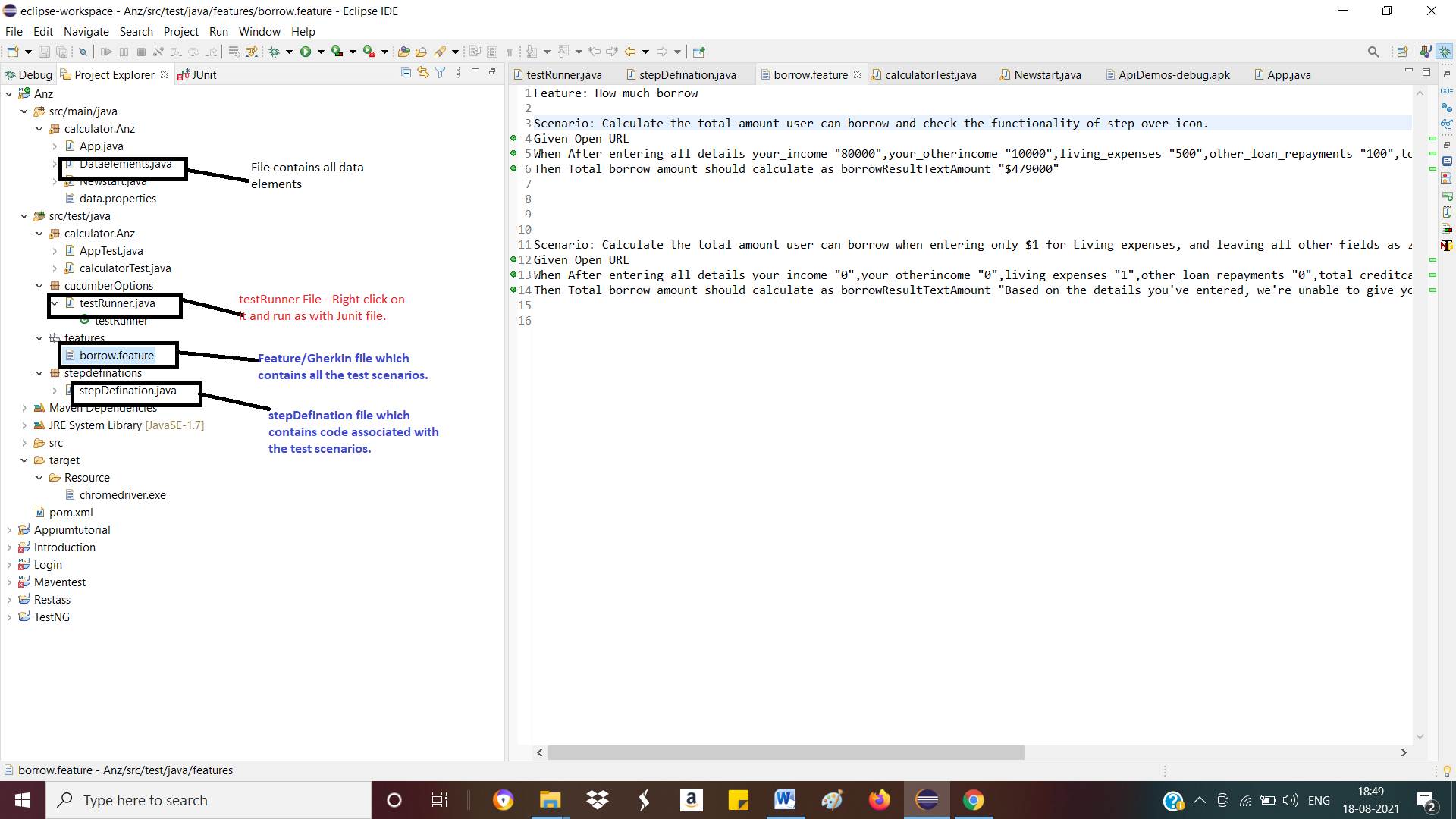
The project should have clear instructions on how to build and run tests and to specify where to find results.

**Questions**

1. What other tests would you suggest could be written? You do **not** need to write them out in detail or code them.
2. If this test was part of a much larger test set, how would you speed it up?
3. Sometimes UI tests can fail unpredictably. For example, the page may not have fully loaded before test automation attempts to click a button on a webpage. How would you improve reliability of these tests?

Good luck!

**Answers:**

****

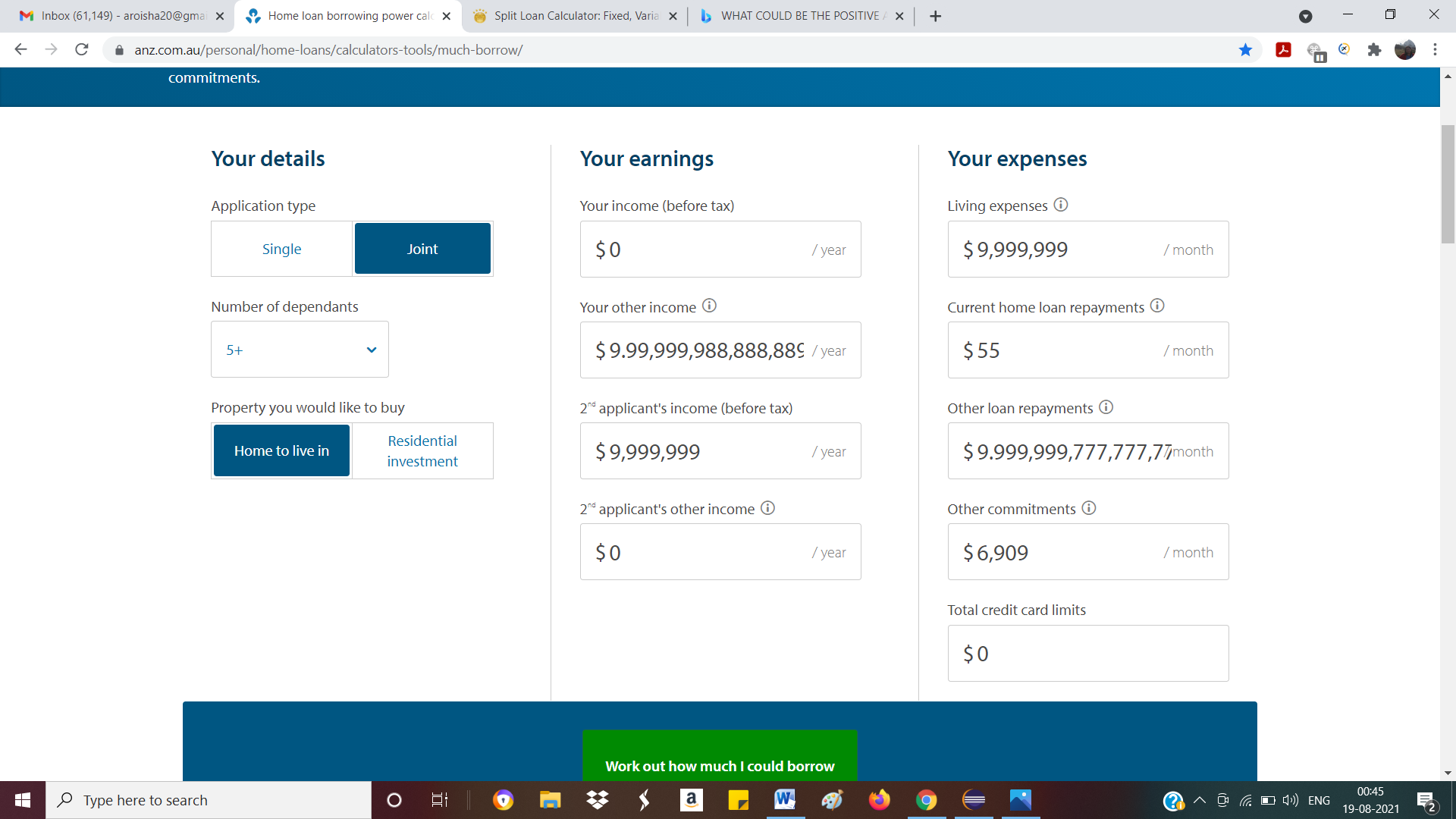
**TestRunner File : Right Click on Test Runner file and Run as with JUNIT Test.**

**Borrow.feature file/Gherkin : This file contains all the test scenarios.**

**Step Definations file: This file contains all the code associated with test scenarios.**

**Test Cases:**

1. **There should be field validations. For eg: Limit on the amount of value . It should not overlap with the field text.**



1. **Total borrow calculated amount should be correct according to the formulas or values entered in all fields.**
2. **Verify the validation message in case of entered incorrect value in the fields.**
3. **Start over icon should appear after disappearing of How much could I borrow button.**
4. **Verify all the API’s for testing the backend process.**

**Ans 2. If this test is the part of much larger test set then we should opt for good framework which minimize our code and all our data set should be separately handled via data driven or parameterized testing.**

**Ans 3. Yes, its true sometimes UI test can fail unpredictly. To improve the reliability of such test, we should introduce the path of timeouts, sleep , webdriver waits into our code so that (driver .find element) statement takes the complete time to find the appropriate element.**

**We can also use the assertions as safety nets so if something goes wrong, it would help you to pinpoint quickly the root cause.**