Ensek Excersize 1: **TESTING APPROACH**

1. Pick an approach: As it’s a web application, I would pick Domain driven testing as this approach bridges the gap between the business orientated stakeholders and developers.

Ideally I would have access to software like testrail ( <https://www.testrail.com/>) which lets you create Test Suites, Plans and Testcases with unique Id’s that can be tracked, and also add these testcases into TestRuns.

As I am on a personal computer I have used Microsoft excel which isn’t perfect but atleast has table formats.

1. Create Test suites: I would come up with all the Initial Test Suites first based on domains. Doman Driven Design is about approaching sections from a business perspective so it would be split by: Users, Buying Energy, Selling Energy, About us page & Contact Us page.
2. Create Sections for each Test Suite: The Test Suites would be split into sections by components (These may match the components created in typescript that can appear on multiple pages). Actions also can be used as Sections of Suites. Here are examples of sections under a specific suite: User -> Login, User -> Registration, Buying Energy -> Buying Gas, Selling Energy -> Selling Oil, Contact us page -> Feedback, etc.
3. Add Test cases to each section: Each section would be filled out with test cases. E.G. Under buying gas, we would detail all the scenarios here. Once done, would double check each webpage to see if there are any use cases left that don’t belong to specific components and these would go under more general page Suites. E.G. I already mentioned an About us page, but we may need a home page Suite just to test the “Find out more button”.
4. Assign priorities to test cases: Possibly as part of the test case creation (or after), we would need to assign priorities to the test cases. Theres two important things to consider here: Business impacts showing how bad for the business it would be if the test was not passing, and a test automation complexity rating (If a test is complicated to automate and low business risk, it may not be worth automating).
5. Create Automated scripts: Start automating the test cases, starting with the Highest Risk ones (And Less complex to automate) for easy wins to get coverage up quicker. Then move on to the other high risk test cases and work our way to the less risky ones if we have enough time.

Ideally these tests would be automated as the developers are creating the features, we would ideally need designs of the page or atleast an overview on how the feature will work, and agree what css UI hooks will be used so we can write our tests. E.G. Make sure a login button has id: LoginButton so the automation code will be able to find it correctly. we can write the automation even when the page itself isn’t complete yet but we could also deploy feature environments to see the in progress work.

1. Create Test Plans – We could do regression test runs of everything, or for new features we could run all the tests in areas that may be impacted by the new feature.
2. Execute Test Plan – Run automation first, and can manually run any tests that aren’t automated after or while automation is running. Then move on to exploratory testing.
3. Exploratory Testing: Used to find bugs that usually aren’t covered in automation tests like spelling issues, design issues, etc which are found easier from human eyes. We may find missed use cases through human error this way and can add them too.
4. Test Results: Check the results of the tests and report and bugs found with a priority rating. We should add details of how to replicate the bug as well as screenshots or short videos. We may also find defects with the test case itself if its outdated so we can rectify the test and retest it.
5. Analyse the test plan: Make sure the tests are still up to date and review the process to see how effective it was / if it needs any adjustments.