This file contains theoretical part of the task and includes initial task in addition to answers (main questions from the task are in bold, answers below questions in regular + several comments in italic)

Basics

● How do you deal with flaky tests?

- *Element waits* - helps me to deal with load specific flakiness - in every method where some visual changes are expected I can wait for some element which means that loading is finished (or wait until I do not see any element).
- *Page waits* - like wait until page fully loaded
- *Test reruns* - adding an ability to rerun the test gives coverage for some unexpected cases (like driver errors for example)
- *Npm installation reruns* - for running test in CI you need to install dependencies, so installation can also use reruns(in my experience with GitHub Actions)
- *Use API calls where possible* - If some entity creation is not important for UI check it can be moved to API call which will decrease possible UI flakiness.
- *Tests are as small and as independent as possible* - general practice to exclude flakiness in advance

● Let's suppose there is a test pipeline taking about 1 hour to finish, what would you do to decrease the time of it?

- Refactor tests if possible - if some steps can be improved, for ex. Use api calls for some actions if applicable; also it is possible to remove some extra verifications or steps details
- Reevaluate needed test set - in case every feature runs the whole test set independently of the changed area, tests may be separated into sets, only related sets should be run and it can decrease the time.
- Prioritize tests - run only top priority tests.

● Imagine you have the possibility to ask software engineers to develop tools for you that will increase your productivity as full-stack QA, please describe to them your requirements

I think a lot of cool tools already created, like automation test frameworks, but I have 2 example ideas which may be helpful and can speedup QA tasks:

1. For test automation - element selector finding tool (most likely browser extension) - similar tools exist in browsers, but selectors are usually long and not perfect. So requirements are next:

- When I open any web-page Then I can switch to the “selector founder” page mode (how to switch mode on can be specified after confirming that it is an extension or build-in browser option)
- When the page is in “selector founder” mode Then I see a sidebar panel with selector editor options and close “selector founder” mode button.
- When I see selector editor panel And do not click on any element Then I see in it help text which says “Please click on any element on the page to see it’s selector”

- When the page is in “selector founder” mode And I click on the element Then I see the full path to the element, recommended selector (short and clear) for it and search element field in the sidebar.
- When I review recommended selector for it Then I can switch selector type xpath/css/class/id/custom
- When I select a selectors type which is not available for the element (for ex., not all elements may have id) Then I see an empty field and help text which says “<selector type> is not available for the element. Please try another selector type”
- When I select a custom selector type Then I can specify an attribute name by which I want to build a selector.
- When I see full path to the element in the dome Then I can copy it, but cannot edit it
- When I see recommended selector Then I can copy selector, but cannot edit it
- When I see search field Then I can paste recommended selector into it and see how many elements are found by it on the page
- When I use search field I can change selector in it and see how many elements are found on the page
- When I search for a selector Then all found elements are highlighted on the page
- When I close “selector founder” mode Then I can interact with the page in regular way
- When I close/open “selector founder” mode Then page state changes quickly and without issues.

2. For bug tracking - voice to text addon in Jira for a full bug creation process (typing the text can take quite some time, so it would be good to have a fully automated process where you can just say and it will create a text from your speech).

- When I click create new defect Then I can select needed field and dictate my text
- When I dictate my text Then I see it properly added to the selected field
- When I want to add some text I can dictate it And see it is added to the field in the place where I set coursore.
- When I add the text by dictation Then I can update it easily in the field
  Here I will also specify all fields in which I want voice to text option, languages which it should support and other possible details

  (Please note: Acceptance criteria are not completely full, they may be even more detailed after design stage and discussion with developers about possibility to implement some points)

Test Case Challenge

NOTE: as this part should take only 30 minutes I will add short version of test plan which will cover only specified questions.

Prepare a test plan for testing the login feature of your application, considering the following requirements:
● The feature is implemented on iOS, Android and Web and is backend driven

● The mobile apps are native
● The login feature consists of a login form with email and password input and a login button
● After input correct credentials, you land in the main screen of your application

Your testing plan need to answer the following questions:

1. What are you going to test?

- Login functionality should be tested on:
  iOS, Android and Web (Chrome, Firefox, Edge and Safari)
- Which types of testing need to be used:
  - Unit and Integration tests (implemented by developers)
  - Functional UI tests (by manual and automation QA)
  - API testing
  - Security testing (security QA + developers)
  - Performance testing (performance QA + developers)
    Two last types depend on the budget and current needs.
- Cases which should be covered:
  - Both fields are empty -> Error
  - Only email: incorrect (I) (depends on validation rules), correct but not registered (CnR), correct registered (CR) -> Error
  - Only password: incorrect (I) (depends on validation rules), correct but not existing (CnE), correct existing (CE) -> Error
  - Login and password pairs (L and P):
    I + I = error
    I + CnE = error
    I + CE = error
    CnR + I = error
    CnR + CnE = error
    CnR + CE = error
    CR + I = error
    CR + CnE = error
    CR + CE (but from different accounts) = error
    CR + CE = success login
  - Error types which may be tested - validation errors, functional errors on button click
  - If the button changes state after input it can be also tested.
  - If security testing in in scope - input fields may be tested using scripts
  - If loading testing is in scope - x simultaneous logins may be tested (same source, i.e. Web or mixed sources - apps + Web)

2. What would you automate and at which layer of the testing pyramid will you place it?

- Unit test (first pyramid level) coverage should be made by developers and ideally 80% code should be covered. Unit tests should run automatically.
- API testing - all possible system answers may be automated (because API testing is quite light and quick) - API testing covers partly integration test level of the pyramid + system level.

- E2E tests - on this level I would automate only first priority tests, because e2e tests are quite slow and should not take too much time. Here I would place a success login path, one of the incorrect login paths.
- In case security and performance tests are involved they may add additional scenarios to each level of the pyramid.

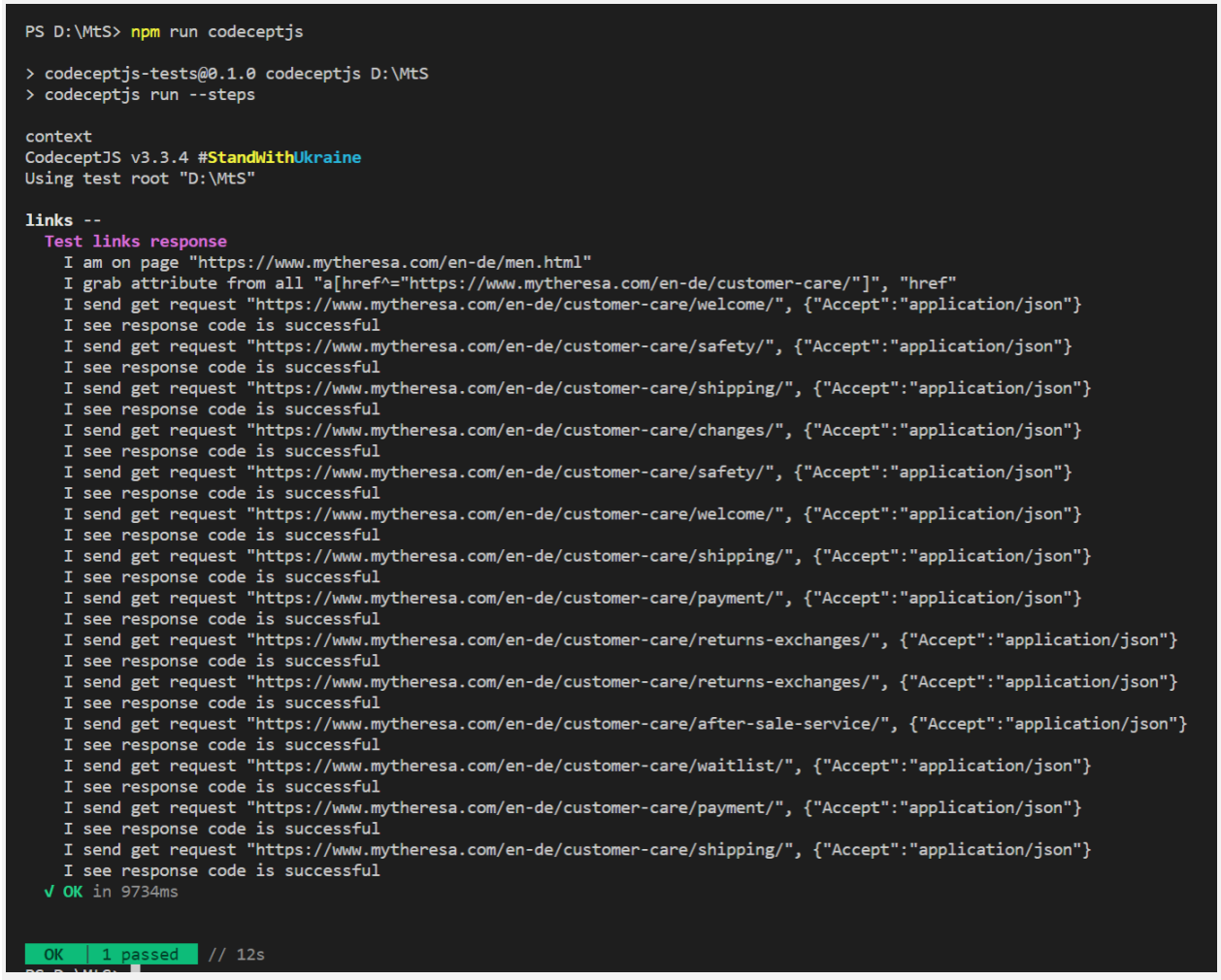
Automation Test Challenge

I'm really sorry, but this task I failed based on acceptance criteria which were added in the task. I never created full codeceptjs/playwright framework from scratch, so when it came to page object part I faced some errors and didn't manage to resolve them during available time slot :(, so was able to create only first test without proper project structure and for limited amount of links (because total links count was 1858 and took to long to run)

But I still decided to upload it to the GitHub for the reference:
https://github.com/DashaMe/MtS

Dependencies were installed using terminal command:
`npm install --save-dev codeceptjs playwright typescript ts-node`

After that you should be able to run tests using
`npm run codeceptjs`



Regarding 2 last tests I may say that I didn't have a chance to try the automation and I did not have similar cases in my experience, but I will provide basic steps how I will try to resolve these problems:

2. Test Case

● As a tester owner, I want to verify I can log in to https://www.mytheresa.com/ende/men.html. Hint: you can use https://maildrop.cc/ to create an account on the fly.

1. Create data using different text generators to create random data for sign up
2. Sign up
3. Trigger mail sending if possible
4. Visit mailbox and check received email text

3. Test Case

● As a product owner, I want to see how many open pull requests are there for our product. You can use https://github.com/appwrite/appwrite/pulls as an example product ● Output is a list of PR in CSV format with PR name, created date and author

1. Grab all needed data from pull requests and store it in the lists or dictionaries
2. Trigger file creation using gathered data
3. Save file