Test Plan Documentation – Free choice group

HomeDork – Interactive Smart House

Revision History

Name	Associated Letter
Fanny Söderlund	A
Malek Alabed	B
Nishat Jahan	C
Suzanne Zomer	D
Ismail Eyamba	E

Date	Version	Description	Author
23/10/2021	1.0	No testing yet currently in concept state.	A, B, C, D, E
27/10/2021	1.1.0	Grammar revised	E
14/11/2021	1.1.1	Changes in document formatting such as versioning, tables, and titles according to group standards.	A
14/11/2021	1.1.2	Addition of episodes Firebase test and Github workflow. Cypress tests on game (test ID 1, 2).	A
4/12/2021	1.2.0	Discuss tests	A, B, C, D, E
7/12/2021	1.2.1	Addition of tests related to disco and voice to text	A

Implemented integration test cases

Test case no	Summary	Precondition	Steps	Results	Comment (Timeline tracked)
1	Check if game loads	Tomcat server is running	- Visit URL - Check if the page contains an element called ‘game’	PASS	Cypress test
2	Check if character and stone image is present on the page	Tomcat server is running	- Get the div with the images - Find elements of ‘img’ - Assure they are visible	PASS	Cypress test
3	Check if disco lights are turned on when pressing button	Tomcat server is running	- Visit the disco page - Acquire the disco button - Press the disco button - Assure the lamps are visible	PASS	Cypress
5	Check if voice to text feature listens to mic	Tomcat server is running	- Visit URL - Get body, then voice button - Click voice to text button and assure the button reacts to method “voice to text”	PASS	Cypress
6	Check if mood commands are visible	Tomcat server is running	- Visit URL - Find href and click it. - Assure URL is status.jsp and acquire mood rectangle divs	PASS	Cypress

Automation tests

Firebase robo test

In the system, we use Firebase as an authentication method for the users. This because firebase provides a good and secure feature for relating passwords to users. We have also decided together with the Units group to use firebase for some of our features such as [R5 – Scheduled commands]. Other features will use the local storage on the device, but that is more discussed in requirements document.
Firebase has a function where they perform robotic tests on an application. This robotic test mainly tests the user interface and can give results on how easy it is to press a button or how well the text is displayed. This robo test gives a full report on the application as a whole and can also test the relation of the app to the firebase storage. However, we will mainly use it for testing the user interface and relate to our supplementary requirement of [S1 - Usability, easy to use].

The robo test is very easy to use, it uses a crawler to check each corner of the application and records the activity and response from the app. The crawler records the response time and will also test for unwanted user activities by acting as a bad user and test the strength and durability of the application. The robo test has support for both Andorid and IOS and we can easily test our android mock app by generating an APK file and uploading to Firebase.

The result of an initial test is promising which includes [R1 – Haptic vibration] and [R8 – Game]. For each couple features implemented, a new robo test can be run since it tests the whole application and not just a single feature.
Figure 1 shows how the crawler works through the app and the different screens.

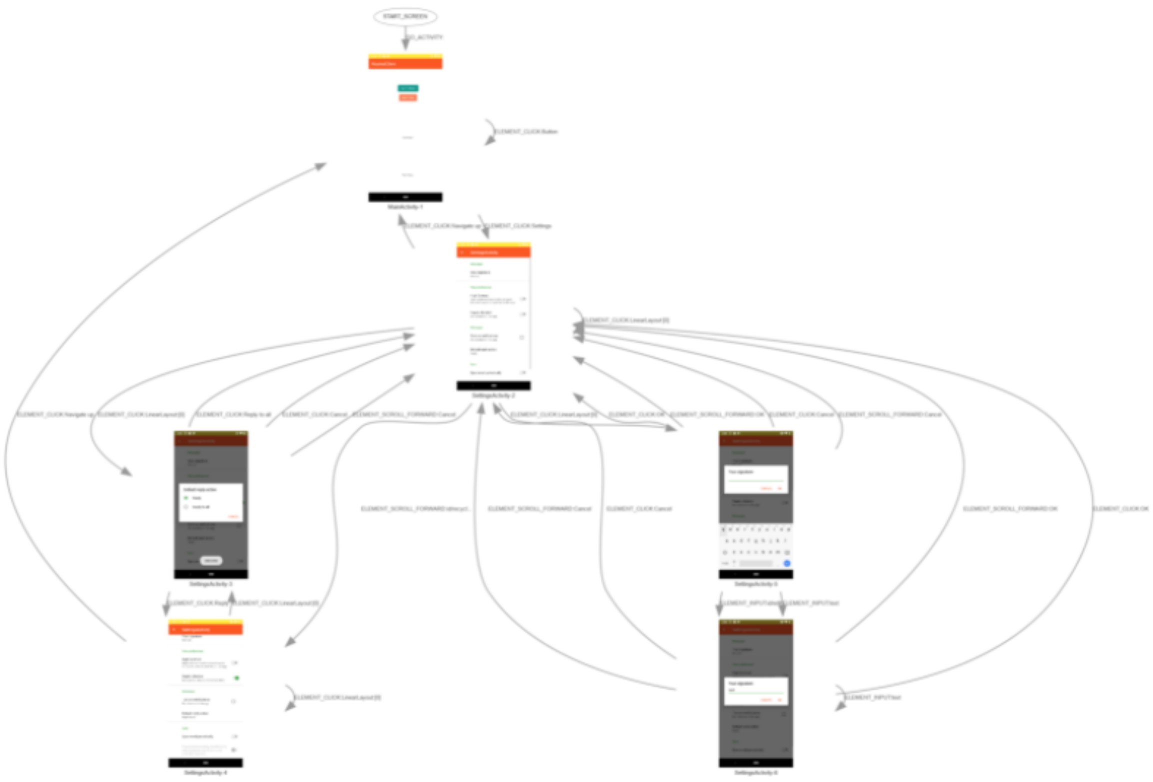
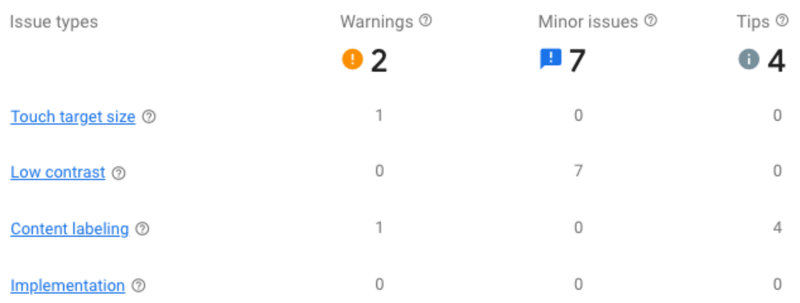


Figure 1 - Crawler work of robo test

The test passed with 2 warnings and 7 minor issues. The 2 warnings are about the touch size for accessibility, however, both are for buttons we have used to access the actual features in our mock environment, so they will not be implemented in the actual system.
Figure 2 shows a complete overview of the test results regarding user interface and accessibility.

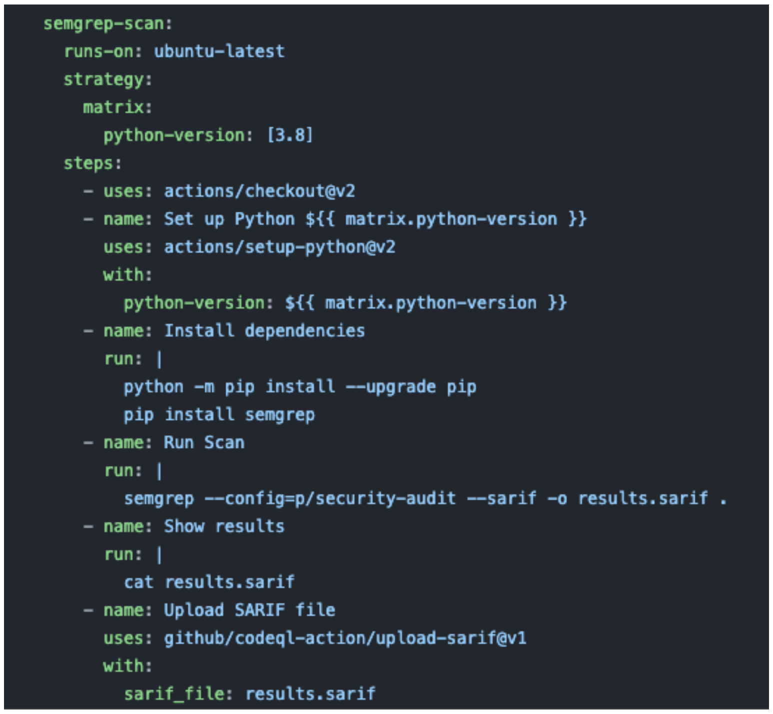


Issue types	Warnings 2	Minor issues 7	Tips 4
Touch target size	1	0	0
Low contrast	0	7	0
Content labeling	1	0	4
Implementation	0	0	0

Figure 2 - Overview of test results

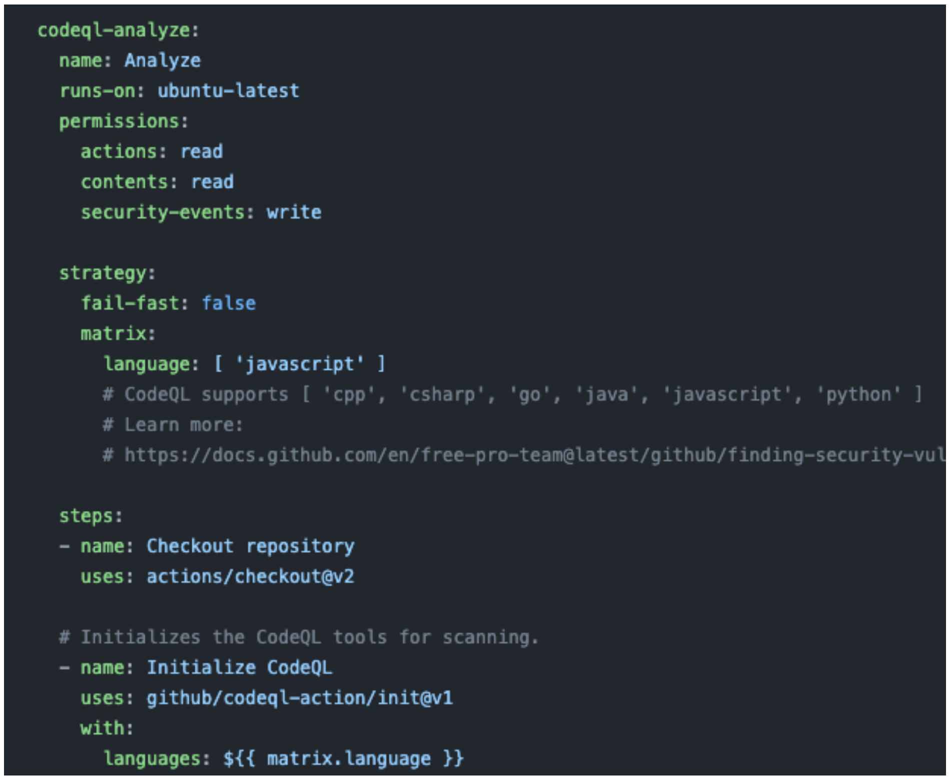
Github workflow

In the HomeDork organization, there is set up a workflow for GitHub actions that are accessible for each repository in the organization to adapt as a template. This workflow is using different security scanning tools to check the overall security of the application. This is especially interesting for our JavaScript code which can be very insecure if not handled correctly. We mainly use the semgrep scanning tool which uses Python and can check most languages. The workflow is run for each pull request through GitHub actions and will deliver a .sarif file with the information about the security level. Figure 3 shows the YAML for this job. This security scanning workflow also has CodeQL set up as a job. CodeQL is a very popular scanning tool using a query language. The query is already set up in the workflow and all that needs to be done in each pull request is to change the language used for the code snippet added. Figure 4 shows a part of the YAML of the CodeQL.
Another action implemented is a check for dependencies and their updates. A deprecated dependency can lower the overall security and for a malicious user, there needs to be only one hole in the system.



```yaml
  semgrep-scan:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.8]
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python ${{ matrix.python-version }}
        uses: actions/setup-python@v2
        with:
          python-version: ${{ matrix.python-version }}
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install semgrep
      - name: Run Scan
        run: |
          semgrep --config=p/security-audit --sarif -o results.sarif .
      - name: Show results
        run: |
          cat results.sarif
      - name: Upload SARIF file
        uses: github/codeql-action/upload-sarif@v1
        with:
          sarif_file: results.sarif
```

Figure 3 - Semgrep job on security workflow



```yaml
codeql-analyze:
  name: Analyze
  runs-on: ubuntu-latest
  permissions:
    actions: read
    contents: read
    security-events: write

  strategy:
    fail-fast: false
    matrix:
      language: [ 'javascript' ]
      # CodeQL supports [ 'cpp', 'csharp', 'go', 'java', 'javascript', 'python' ]
      # Learn more:
      # https://docs.github.com/en/free-pro-team@latest/github/finding-security-vul

  steps:
  - name: Checkout repository
    uses: actions/checkout@v2

  # Initializes the CodeQL tools for scanning.
  - name: Initialize CodeQL
    uses: github/codeql-action/init@v1
    with:
      languages: ${{ matrix.language }}
```

Figure 4 - CodeQL job on security workflow