How to run a test on your visualization system

folder *material/testing-environment*

(A) Steps to run a test on your visualization system (design or evaluation phase) for the first time:

1. Install the requirements running:

   ```
   pip install -r requirements.txt
   ```

2. Create a virtual environment by running the command:

   ```
   python -m venv <name_of_the_virtual_environment>
   ```

3. Activate the virtual environment by running:

   ```
   <name_of_the_virtual_environment>\Scripts\activate
   ```

4. If an error appears in the console telling that the execution of Scripts is not allowed, open a Windows Powershell terminal in admin mode and enter the command:

   ```
   Set-ExecutionPolicy unrestricted
   ```

   This will allow the policy of scripts execution in your machine to be unrestricted, so be careful

5. In the terminal, execute one of the two commands:

   ```
   set FLASK_APP=server_flask.py
   ```

   ```
   $env:FLASK_APP = "server_flask"
   ```

6. In the terminal, execute the command:

   ```
   flask run
   ```

The server should now be running in localhost. Press the key combination "CTRL+C" to interrupt it. If it is not the first time running the experiment, follow only from point 3) to point 6).

(B) To change the visualization system and to switch from evaluation to design mode and viceversa:

1. In the "static" folder, you can find the file "stein-config.json". Change the "systemUrl" value to the desired url.
2. In the "server_flask.py" file, in row 18, change "evaluation.html" in "design.html". This is needed in order to design the new experiment. Remember to revert this change, when the design phase is finished, in order to run the real evaluation.
3. Follow all the steps of (A) if it is the first time running the design OR evaluation phase, otherwise follow only from point 3) to 6) of A)

4. The design phase should be running in localhost
5. At the end of the design phase, a new "stein-config.json" can be downloaded. Remember to place it in the "static" folder, overwriting the previous one.
6. To run the real experiment instead of the design phase, change, in the "server_flask.py" file, at row 18, "design.html" in "evaluation.html"

N.B: The "Upload" functionality does not work at the end of evaluation mode. Instead, the file, containing the evaluation's results, can be downloaded to your machine.

How to run SWComponent1 (Statechart Generator)
folder material/statechart-generator

In order to build and run it on your machine, you must have already installed and configured:

- NodeJS v12.20.2
- Puppeteer node module v13.7.0
- fs node module v1.0.0
- is-same-origin node module v0.0.7

Then you can open a terminal inside that folder and run the *main.js* file with NodeJS. You can specify the link to the visualization inside the *./material/system_url.txt* configuration file, while the list of excluded events can be customized inside the *./material/excluded_events.txt* configuration file.

How to run SWComponent2 (Traces Replayer)
folder *material/traces-replayer*

1. Install the requirements running:

   pip install -r requirements.txt

2. Download Chrome Webdriver available at: https://chromedriver.chromium.org/home.
3. Add the downloaded webdriver to system PATH.
4. Change value of url, visualization name and sibling percentage in the **conf.json** file.
5. Run the *PathsSimulator.py* file using a terminal or an IDE like VSCode.

Now you can verify the latencies in the files inside the **resultExplorations** folder, for each interface there will be a file with only the violations (**summaryProblems**) and one with the latency times for each event (**summary**).

How to reproduce the user study
folder *validation/reproducing-the-experiment*

1. Generate the complete statechart of Falcon 7M using the Statechart Generator on this URL https://vega.github.io/falcon/flights-duckdb/. Otherwise you can use the already generated one which is stored in two files, "statechart.json" and "statechart_graphviz.gv".
2. You can easily visualize the "statechart_graphviz.gv" file by processing it in https://dreampuf.github.io/GraphvizOnline/ which converts it in a .svg file, in order to manipulate the edges and nodes to make them more visible. The two state charts in the paper have been coloured considering the number of times a specific interaction has been recorded (first state chart) and considering the violations in latency thresholds recorded (second state chart).
3. In the count folder:
    a. The count.py script opens a single exploration file and returns as output a file named "count_falcon_7M_1.json" (where 1 is the number of the exploration) that contains the count of all the interactions recorded, divided by task. Also, the script updates the .xlsx files that are divided by task (task number 1 is the tutorial that the users needed to complete before proceeding with the other tasks). The other "summary_falcon_7M.json" files are the explorations that the count.py script actually uses as input.
4. In the times_and_violations folder:
    a. The times.py script opens a single exploration file and returns as output a file named "violationsFound_7M_1.json" (where 1 is the number of the exploration) that contains all the violations in the latency thresholds that were found, divided by task. The script updates the .xlsx files with the times needed to complete each interaction. The other "summary_falcon_7M.json" files are the explorations that the count.py script actually uses as input (that are different from the ones used by the count.py script). It is possible that some violation output file contains the same violation, but repeated many times; in this case, these violations have been counted as a single violation.