BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

SESSION REPLAY USING AUTOMATED WEB BROWSERS

OPAKOVÁNÍ SEZENÍ S POMOCÍ AUTOMATIZOVANÝCH PROHLÍŽEČŮ

TERM PROJECT
SEMESTRÁLNÍ PROJEKT

AUTHOR **MARKO POĽANSKÝ**
AUTOR PRÁCE

SUPERVISOR **Ing. LIBOR POLČÁK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2025

Abstract

This work deals with the design and implementation of a system for dynamic analysis with session replay. The Puppeteer library was chosen as a suitable library. The implementation enables a dynamic analysis where the output is data about the visiting page such as: page loading time, screenshot, waterfall analysis. The analysis can be performed on the single visited page or by providing a recording of the user's session from the Google Chrome browser, it is possible to perform an analysis on the sequence of recorded events (clicking on a link, filling out a form)

Abstrakt

Táto práca sa zaoberá návrhom a implementáciou systému na dynamickú analýzu s opakovaním sedenia. Ako vhodná knižnica bola vybratá knižnica Puppeteer. Implementácia následne umožňuje vykonávať dynamickú analýzu, kde jej výstupom sú dáta o navštívejej stránke ako: čas načítania stránky, snímka obrazovky, waterfall analysis (dáta jednotlivých zdrojov stránky). Analýza sa môže vykonať na samostatne navštívenú stránku alebo poskytnutím nahrávky relácie užívateľa z prehliadača Google Chrome je následne možné vykonať analýzu na sekvenciu nahraných udalostí (kliknutie na odkaz, vyplnenie formulára)

Keywords

dynamic analysis , Puppeteer, session recording, single visited page

Klíčová slova

dynamická analýza, Puppeteer, nahrávka sedenia, samostatne navštívená stránka

Reference

POĽANSKÝ, Marko. *Session replay using automated web browsers.* Brno, 2025. Term project. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Libor Polčák, Ph.D.

Session replay using automated web browsers

Declaration

Prehlasujem, že som túto semestrálnu prácu vypracoval samostatne pod vedením Ing. Libora Polčáka Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Marko Poľanský
February 25, 2025

Contents

1 Introduction **3**

2 Selection of a Suitable Library **4**

3 Session Recording **5**
3.1 Obtaining the Session Recording 5
3.2 Types of Recording Steps 7
3.3 Using the Timeout Property 8
3.4 Large Number of Steps 8
3.5 Mouse Movement 8
3.6 Cookies 8
3.7 Selector Matching an Unintended Element 8

4 Implementation **9**
4.1 Data Flow Diagram 9
4.2 Single Visited Page 10
4.3 Recording of Multiple Page Visits 10
4.4 Output 11
4.5 Potential Security Issues 12

5 Testing of a Single Visited Page **13**
5.1 Case of Dynamically Changing Pages 13
5.2 Case of Fully Loaded Page After Load Event 14

6 Testing the Recording of Multiple Visited Pages **15**
6.1 Case of Overlaying Window 15
6.2 Case of Login 17
6.3 Case of page_changed 18
6.4 Case of Dynamically Changing Pages 18

7 Conclusion **19**

Bibliography **20**

List of Figures

3.1 More options → More tools 5
3.2 Create a new recording 6
3.3 Start recording 6
3.4 End recording 6
3.5 Puppeteer 7

4.1 Data flow diagram 9

5.1 A different `screenshot_timing_type` was selected for each analysis. Starting from the left, `load`, `networkidle0`, and `networkidle2`. 13
5.2 For each analysis, `screenshot_timing_type load` was selected, and starting from the left, `screenshot_timing` was set to `200`, `1000`, and `5000` milliseconds. 14
5.3 A different `screenshot_timing_type` was selected for each analysis. Starting from the left, `load`, `networkidle0`, and `networkidle2`. 14

6.1 The red rectangle indicates the covered element that was supposed to be clicked. 16
6.2 10 recorded steps, 9 of which are part of the output. 17
6.3 Page at the beginning and end of the recording 17
6.4 The pages are not fully loaded 18
6.5 The pages took longer to load 18

Chapter 1

Introduction

This work is part of the overall INVENTOR project by Flowmon Networks in collaboration with the Brno University of Technology (VUT). The INVENTOR project focuses on combined passive and active network monitoring. The work on the INVENTOR project was divided into four phases. The VUT team worked on the first phase: „Development of a Monitoring Agent.“ Each member of the VUT team was responsible for a monitoring agent for a specific network layer service. In this work, it was necessary to consider the selection of a suitable library that would allow obtaining the necessary data from a visited website while also enabling the playback of a recorded user session. The section on session recording describes how such a recording can be obtained, what it may contain, and its limitations. Subsequently, the implementation is described for both an individually visited website and a recording of multiple visited websites. At the end of the work, a comparison of the output of dynamic analysis in various website visitation scenarios is presented.

Chapter 2

Selection of a Suitable Library

`Flowmon Networks` documentation recommended Puppeteer, but after comparison, switching libraries was possible. For dynamic analysis of the HTTP protocol, the library should allow obtaining the following data:

- Browser processing time (DOMContentLoaded and load)
- Size of downloaded data
- Waterfall analysis[1]
- Screenshot (Base64)

Other requirements:

- When replaying a session, it is necessary to consider whether the library allows both recording and replaying the session.
- The programming language chosen for implementing the monitors was Python. Compatibility of the library with Python would allow more efficient data manipulation.

Library	**Processing Time**	**Data Size**	**Waterfall Analysis**	**Screenshot**	**Python Support**
Puppeteer	Yes	Yes	Yes	Yes	No
Selenium	Yes	Yes	Yes	Yes	Yes
Playwright	Yes	Yes	Yes	Yes	Yes

Table 2.1: Comparison of data extraction support and Python compatibility

Library	**Session Recording**	**Session Playback**
Puppeteer	Built into Chrome	Yes
Selenium	Chrome Extension	Yes
Playwright	Visual Studio Extension	Yes

Table 2.2: Comparison of session recording and playback

The `Puppeteer` library meets all the necessary conditions while providing the simplest and most intuitive way to obtain recordings. Moreover, the use of the `Puppeteer` library was recommended by `Flowmon Networks`. Based on these reasons, the `Puppeteer` library was selected.

[1] For all resources of the visited page, extract the request method, status code, size of the resource, start time of resource download, and total resource download time

Chapter 3

Session Recording

The session recording for the `Puppeteer` library can be obtained from the Google Chrome browser. The recording contains individual steps chronologically recorded based on the user's interactions with the web.

3.1 Obtaining the Session Recording

1. Open `DevTools` in Google Chrome.
2. Click on `More options` → `More tools` → `Recorder`.

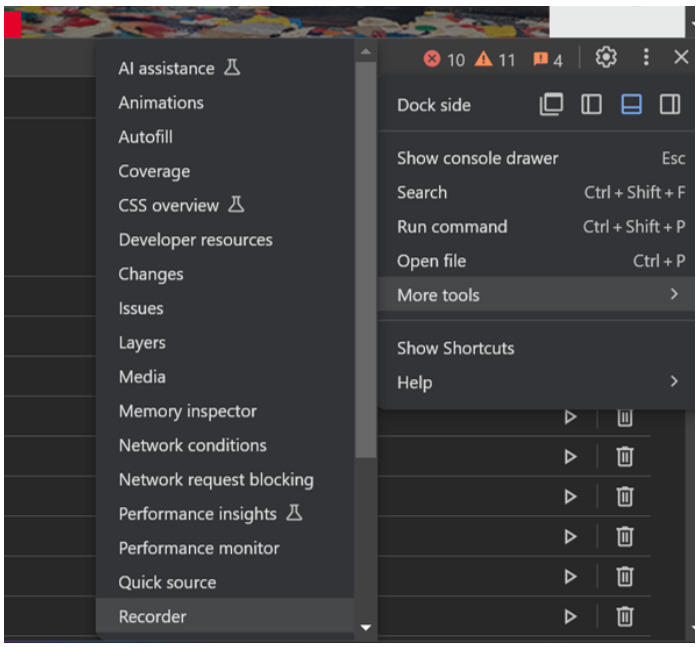

Figure 3.1: More options → More tools

3 Click on `Create a new recording` → `Start recording`.

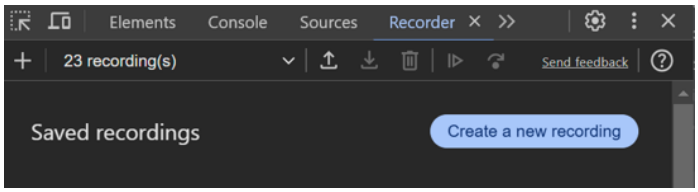

Figure 3.2: Create a new recording

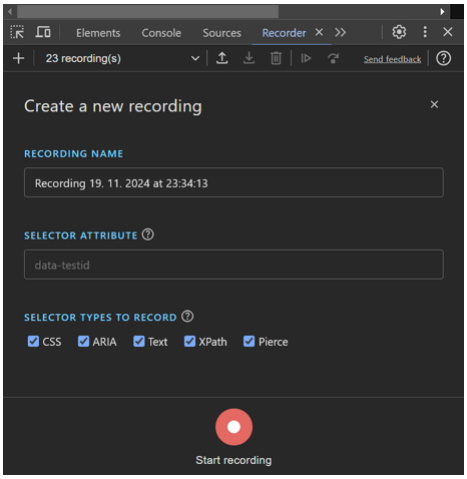

Figure 3.3: Start recording

4 Click on `End recording` → `export` → `Puppeteer`.

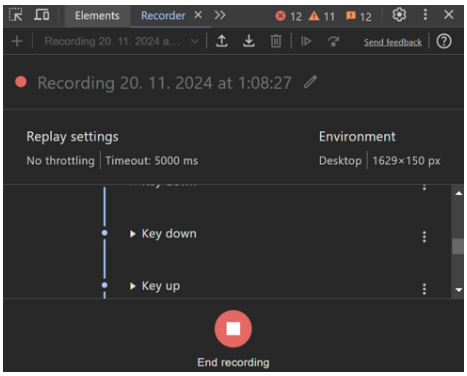

Figure 3.4: End recording

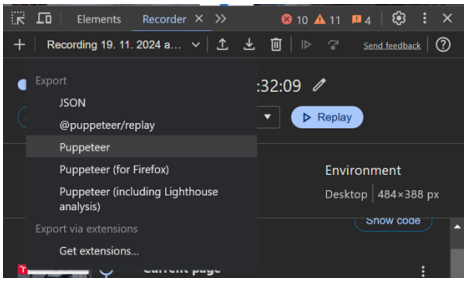

Figure 3.5: Puppeteer

3.2 Types of Recording Steps

A recording from the Google Chrome browser consists of multiple steps, which are then translated into the selected language and library upon export. The documentation [2] provides the following options for selecting steps and additional settings. Not all steps are included in the recording by default, some must be added manually. The `mandatory addition` field indicates that certain steps appear in the recording only if manually added.

Step	**Description**	**Mandatory Addition**
click	Clicking on an element	No
doubleClick	Double-clicking on an element	No
hover	Moving the cursor over an element	Yes
(input) change	Changing the value of an input field	No
keyUp	Releasing a key	No
keyDown	Pressing a key	No
scroll	Scrolling a page or element	No
close	Closing a page or dialog window	No
navigate (to a page)	Navigating to another page	No
customStep	Custom step	Yes
emulateNetworkConditions	Simulating network conditions	Yes
waitForElement	Waiting for an element to appear	Yes
waitForExpression	Waiting for a condition to be met	Yes
setViewport	Setting the viewport size	No

Table 3.1: Comparison of Recording Step Types

Each step may have certain properties.

Property	**Description**
target	URL Chrome DevTools Protocol (CDP) target
assertedEvents	May contain only one navigation event
frame	Array of values determining nested iframes
timeout	Number of milliseconds to wait before executing a step
selectors	Array of selectors for identifying elements

Table 3.2: Properties Available for Most Step Types

3.3 Using the Timeout Property

The `element_search_timeout` value in the monitor implementation serves the purpose of the timeout property. It is then set for all steps, and if this timeout is exceeded, it is assumed that the page has changed. If the timeout property is added for a specific step during recording, the `element_search_timeout` value will be ignored.

3.4 Large Number of Steps

In the monitor implementation, analysis is performed on each step of the recording. Each step will have a record with data and a screenshot of the respective page. This approach allows visual changes and changes in the waterfall analysis on the same page. If a recording with many unnecessary steps (interactions) were created, such as 20 consecutive clicks on an insignificant spot, it would lead to a long execution time and a resulting file containing 20 identical screenshots. Therefore, it is recommended to record only necessary actions.

3.5 Mouse Movement

Recordings do not include mouse movements, which can cause problems. For example, if a menu appears only after moving the mouse and then clicking on a link, the recording will not be able to locate the link, leading to an error. After consultation with `Flowmon Networks`, it was decided to ignore this issue. However, this problem can be solved in two ways:

1. The recording allows editing, where a `hover` step can be manually added to expand the necessary menu.
2. Using the `click` and `focus` events to expand such menus. These events are already included in the recording.

3.6 Cookies

Recordings do not contain `cookies`. If a user records a session where functionality or page structure differs due to `cookies`, playing back such a recording could lead to non-functionality. This issue was discussed with `Flowmon Networks`, and it was decided to ignore this limitation.

3.7 Selector Matching an Unintended Element

If a recorded selector matches an unintended element due to changes on the page, such as a login input being replaced by a search bar, the test would be considered successful even though the page has changed. After consultation with `Flowmon Networks`, it was also decided to ignore this issue.

Chapter 4

Implementation

The result of the work enables dynamic analysis of the HTTP protocol on a single visited page or, by providing a session recording, the analysis of several pages can be performed. The output is in the same format for both types of analysis cases, differing only in the input. If input data is provided for both types of analyses, the analysis for the recording of multiple page visits is performed.

4.1 Data Flow Diagram

The main program runs in `monitor_webapp_dynamic_analysis.py`, which then launches a Node.js script that performs dynamic analysis using the `Puppeteer` library. The result is then sent back and saved to a file in `json` format.

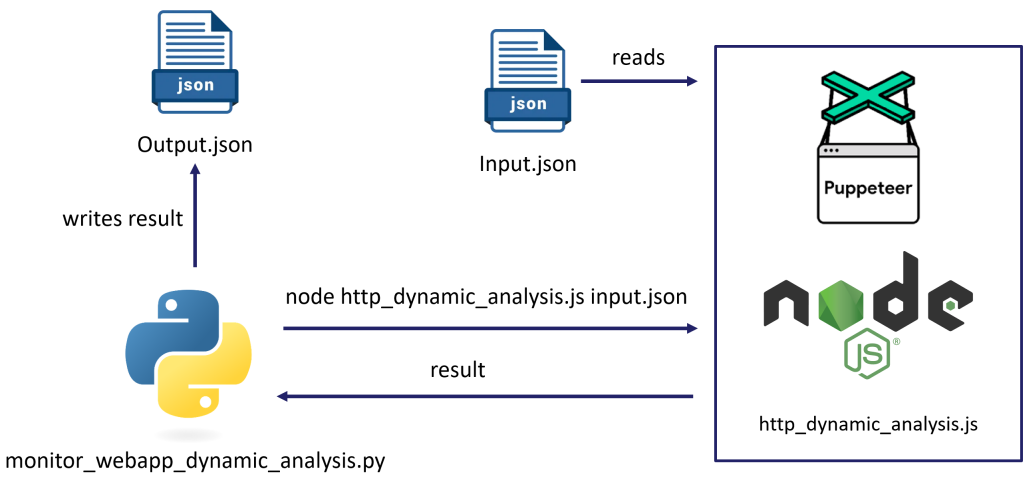

Figure 4.1: Data flow diagram

4.2 Single Visited Page

A single visited page is a type of analysis where the analysis is performed on the URL of a single page.

Input

Parameter	Type	Default	Required	Description
`target_host`	`str`	N/A	Yes	URL address for the HTTP request.
`screenshot_timing_type`	`str`	`load`	No	Screenshot will be taken after an event (`load`, `networkidle0`, `networkidle2`)
`screenshot_timing`	`int`	0	No	Additional waiting time in milliseconds after the `screenshot_timing_type` event.
`timeout`	`int`	30000	No	Maximum runtime of the monitor in milliseconds.

Table 4.1: Input for single visited page

- `load` waits for the `load` event.
- `networkidle0` waits until there are no more than 0 network connections for 500 ms.
- `networkidle2` waits until there are no more than 2 network connections for 500 ms.

4.3 Recording of Multiple Page Visits

The analysis is performed on individual steps present in the recording.

Input

Parameter	Type	Default	Required	Description
`replay_script`	`str` (Base64)	N/A	Yes	Downloaded user session script in `Base64` from Google Chrome.
`timing_between_steps`	`int`	200	No	Waiting time between individual steps.
`timeout`	`int`	30000	No	Maximum runtime of the monitor in milliseconds.
`element_search_timeout`	`int`	5000	No	Time limit for finding an element. Exceeding this indicates a change on the page.

Table 4.2: Input for recording of multiple page visits

4.4 Output

The output format is the same for both single visited pages and recordings of multiple page visits.

Main Parameters

Parameter	Type	Description
`run_id`	`int`	ID of the current analysis run.
`status`	`string`	`completed`, `error`, `page_changed`, `partial`
`error_msg`	`string`	Error message. If `status` is `completed`, this parameter is not present.
`visited_pages`	`list<dict>`	List of visited pages.

Table 4.3: Output main parameters

Items of visited_pages

For single visited page analysis, `visited_pages` contains only one entry. For multiple page visit recordings, it contains individual steps.

Parameter	Type	Description
`current_url`	`string`	Currently visited URL address.
`status`	`string`	`completed`, `partial`, `page_changed`
`load_time`	`int`	Time of the `load` event in milliseconds.
`dom_content_loaded_time`	`int`	Time of the `DOMContentLoaded` event in milliseconds.
`total_data_size`	`int`	Total size of resources in bytes.
`waterfall_analysis`	`list<dict>`	List of network requests.
`image`	`string`	Image in `Base64`.

Table 4.4: Output items of visited_pages

- `partial`: This particular visited page is incomplete due to an error.
- `page_changed`: This particular visited page does not contain the element that the recording interacted with.

waterfall_analysis Items

Each visited step/page contains a list of network requests.

Parameter	Type	Description
`url`	`str`	URL of the requested resource
`method`	`str`	HTTP method used in the request
`status_code`	`int`	HTTP status code of the response
`size`	`int`	Size of the resource in bytes
`start_transfer_time`	`str`	Unix timestamp indicating when the resource transfer started
`time`	`int`	Total duration of the request in milliseconds
`error_msg`	`str`	Error message of the request.

Table 4.5: Output of the waterfall_analysis item

4.5 Potential Security Issues

The user recording is a JavaScript code generated for the `Puppeteer` library. Running such a recording requires executing the code, which could pose a risk if custom code is provided instead of a generated recording. However, this risk is irrelevant in this project since all tests will be performed in the customer's local network, accessible only by the administrator.

Chapter 5

Testing of a Single Visited Page

This section discusses the impact of selected parameters on analyses of single visited pages.

5.1 Case of Dynamically Changing Pages

When analyzing dynamically changing pages, it is necessary to consider that additional data may be loaded after the load event. By appropriately setting `screenshot_timing_type` and `screenshot_timing`, it is possible to ensure that the page is loaded in the desired state. Typically, this might involve single-page applications.

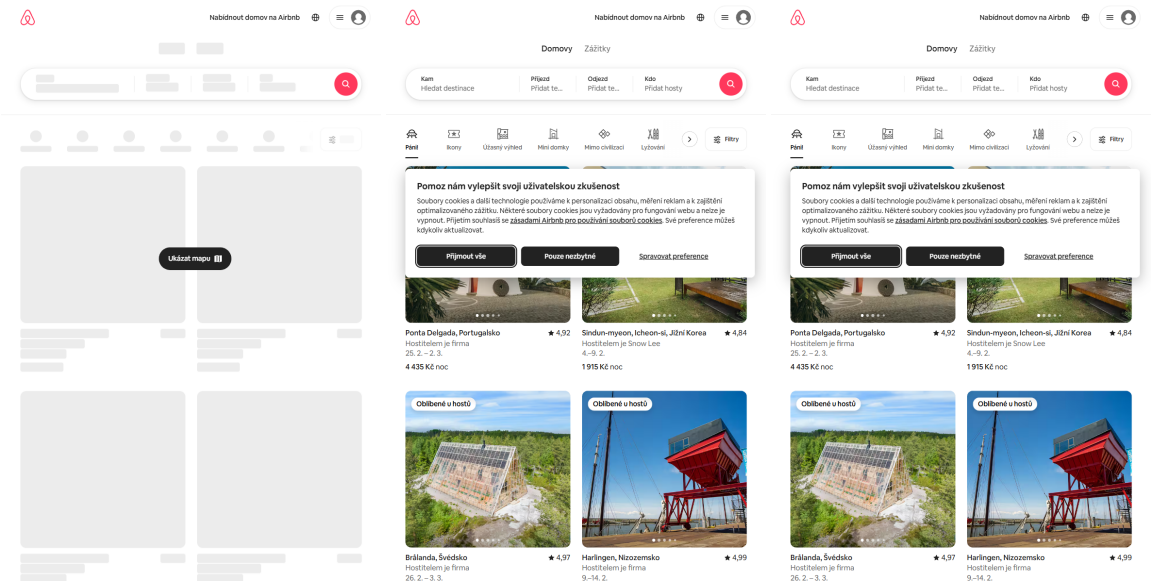

Figure 5.1: A different `screenshot_timing_type` was selected for each analysis. Starting from the left, `load`, `networkidle0`, and `networkidle2`.

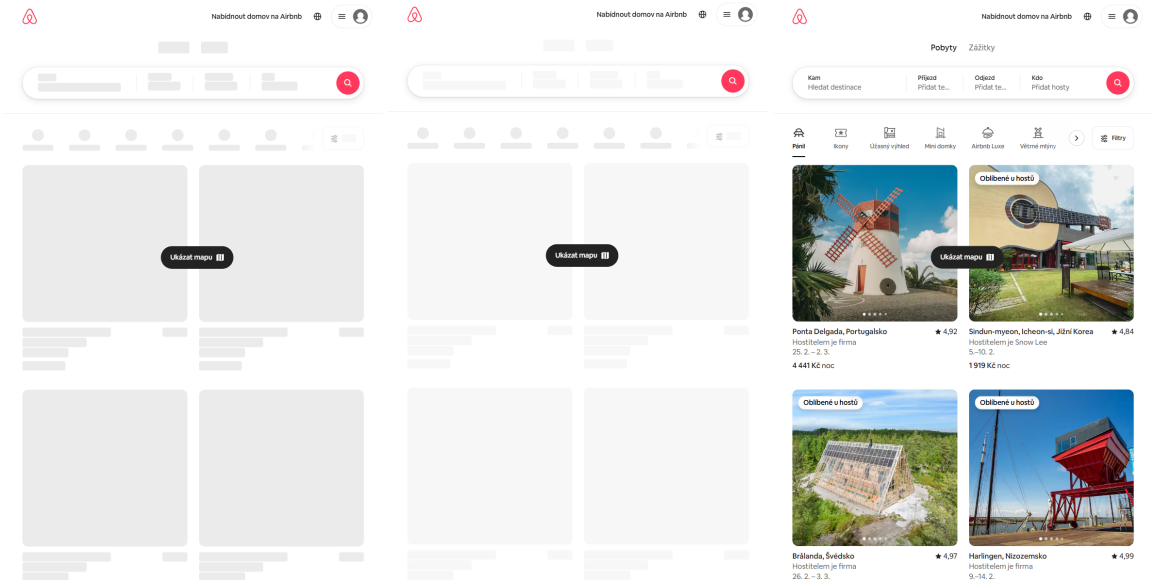

Figure 5.2: For each analysis, `screenshot_timing_type load` was selected, and starting from the left, `screenshot_timing` was set to 200, 1000, and 5000 milliseconds.

5.2 Case of Fully Loaded Page After Load Event

When analyzing a fully loaded page after the load event, there is no need to adjust the `screenshot_timing_type` and `screenshot_timing` parameters. Typically, this might involve multi-page applications. The image shows that the page is fully loaded after the load event.

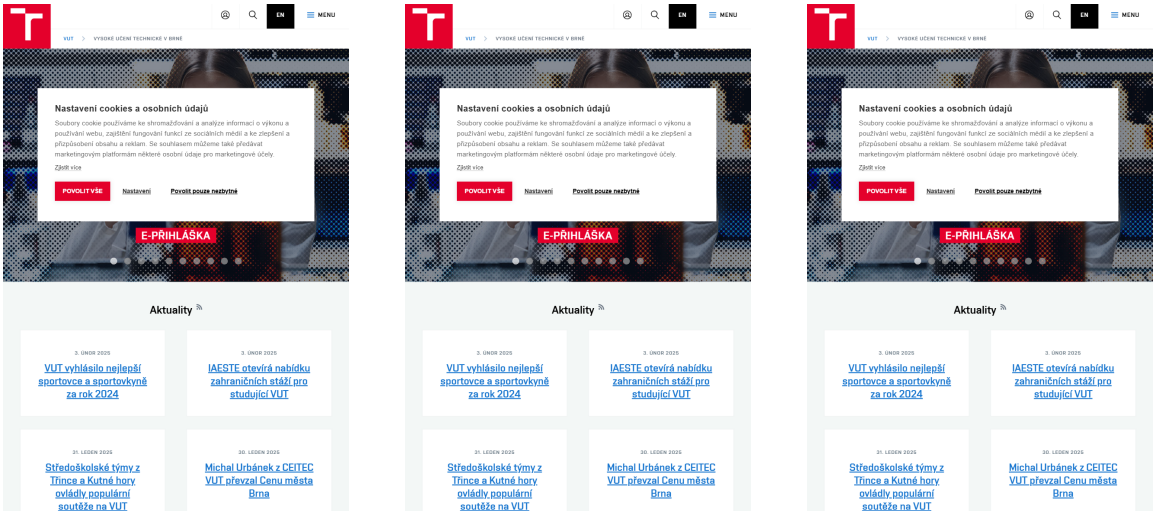

Figure 5.3: A different `screenshot_timing_type` was selected for each analysis. Starting from the left, `load`, `networkidle0`, and `networkidle2`.

Chapter 6

Testing the Recording of Multiple Visited Pages

This section describes how the analysis behaves in various cases when recording the visit of multiple pages.

Cookies

For pages requiring cookie consent (usually via a pop-up), it is recommended to confirm the cookies. The monitor does not retain the state from previous runs, so it is necessary to record the cookie confirmation step in every recording (even for the same website). Possible limitations resulting from ignoring such confirmation:

1. The pop-up window may cover the target element. When clicking on the target element, the click might be intercepted by the window, potentially leading to unexpected behavior.

2. Not confirming cookies may change the functionality and appearance of the page.

6.1 Case of Overlaying Window

Closing the Window

The recording includes clicking the cookie confirmation from the pop-up window and clicking a link leading to another subpage of the same website. The result in this case contains all the expected data.

status	**error_msg**
`completed`	

Table 6.1: Analysis Result

Ignoring the Window

In this case, the recording from the previous example did not capture the closing of the cookie window.

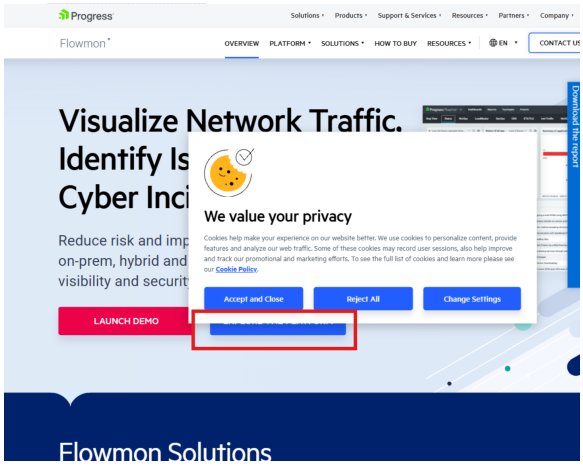

Figure 6.1: The red rectangle indicates the covered element that was supposed to be clicked.

In the original recording, it was expected that clicking the element would redirect to another subpage. Therefore, the recording includes the `targetPage.waitForNavigation()` command. This is executed after clicking the localized element. The function `waitForNavigation()` waits until the page navigates to a new address or reloads [1]. However, the navigation does not occur because the click was not made on the desired element due to the pop-up window covering it.

```
const startWaitingForEvents = () => {
    promises.push(targetPage.waitForNavigation());
}

await puppeteer.Locator.race([
    targetPage.locator('::-p-aria(EXPLORE THE PLATFORM)'),
    targetPage.locator('#Content_C008_Col01 > a'),
    targetPage.locator('::-p-xpath(//*[@id=\\"Content_C008_Col01\\"]/a)'),
    targetPage.locator(':scope >>> #Content_C008_Col01 > a'),
    targetPage.locator('::-p-text(Explore the platform)')
])
    .setTimeout(timeout)
    .on('action', () => startWaitingForEvents())
    .click({
        offset: {
            x: 147.48211669921875,
            y: 26.91070556640625,
        },
    });

await Promise.all(promises);
```

Listing 6.1: Step of Clicking the Element from the Recording

status	error_msg
partial	TimeoutError: Navigation timeout of 30000 ms exceeded

Table 6.2: Analysis Result

The result of such a run is a partial status, indicating that the output data is incomplete due to an error. The error message indicates that a navigation timeout occurred.

6.2 Case of Login

The test recording includes 9 steps (the view step is ignored as the analysis starts after this step). Each step has a record in `visited_pages`. The steps involve actions such as clicking input fields, filling in credentials, and submitting the form.

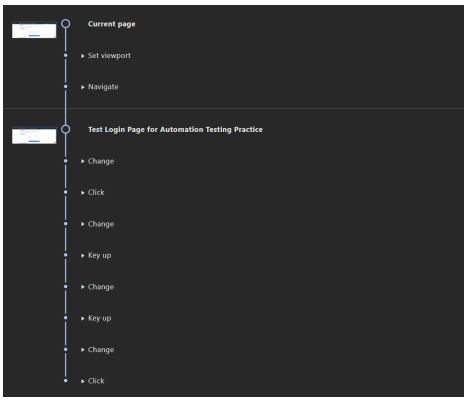

Figure 6.2: 10 recorded steps, 9 of which are part of the output.

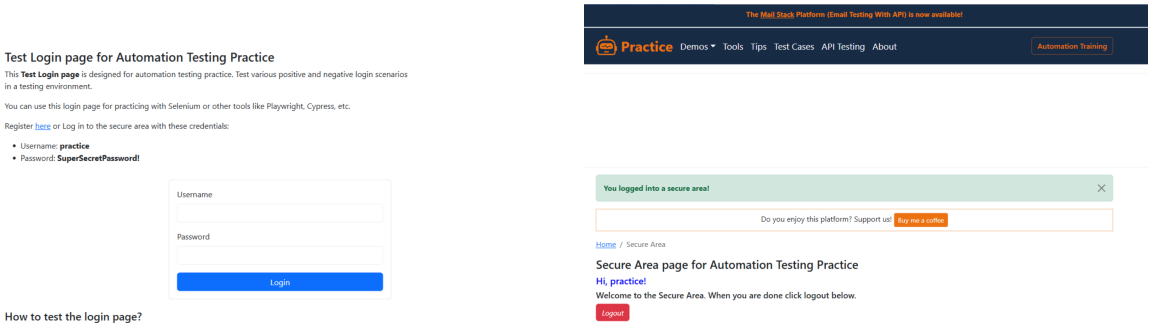

Figure 6.3: Page at the beginning and end of the recording

status	error_msg
completed	

Table 6.3: Analysis Result

6.3 Case of page_changed

This case indicates that the page was changed. The recording includes two clicks: first, closing the cookie window, and second, clicking a banner. The second click used a selector that no longer exists, simulating that the page changed and the recorded selector is no longer locatable.

status	error_msg
page_changed	TimeoutError: Timed out after waiting 5000ms

Table 6.4: Analysis Result

6.4 Case of Dynamically Changing Pages

Similarly to single visited pages, dynamically changing pages can be analyzed. The difference is that the `screenshot_timing_between_step` parameter is used to set the waiting time between steps.

Case of Waiting 200 Milliseconds Between Steps

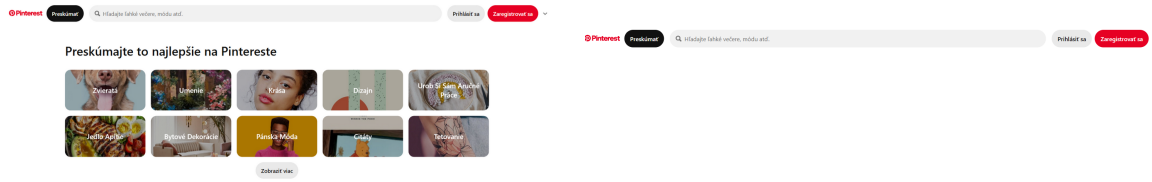

Figure 6.4: The pages are not fully loaded

Case of Waiting 1000 Milliseconds Between Steps

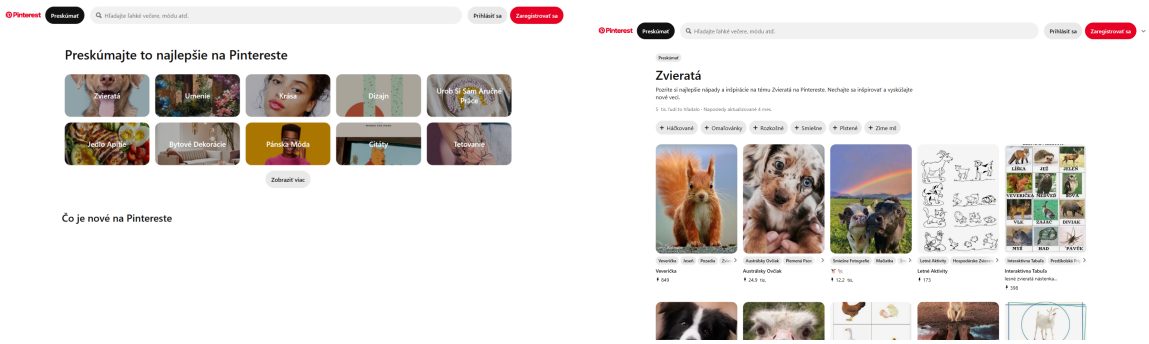

Figure 6.5: The pages took longer to load

Chapter 7

Conclusion

The goal of this project was to create a functional program for dynamic analysis of the HTTP protocol according to the specifications. The project met all the set objectives. The case tests describe various analysis scenarios and demonstrate the program's functionality. However, it is likely that not all possible cases are covered in the test cases, so it is important to pay attention to additional case types in future work. The result of the work is a program with documentation stored in a repository on GitHub.

Bibliography

[1] *Page.waitForNavigation() method | Puppeteer* online. 2025. Available at: https://pptr.dev/api/puppeteer.page.waitfornavigation. [cit. 2025-02-04].

[2] Emelianova, S. and Yeen, J. *Features reference* online. 2024. Available at: https://developer.chrome.com/docs/devtools/recorder/reference. [cit. 2025-02-02].