Bachelor Thesis Project

GUI driven End to End Regression testing with Selenium

Author: Christer Hamberg
Supervisor: Johan Hagelbäck
Semester: VT 2017
Subject: Computer Science

Abstract

Digitalization has changed our world and how we interact with different systems. Desktop applications have more and more been integrated with internet, and the web browser has become the Graphical User Interface (GUI) in today’s system solutions. A change that needs to be considered in the automated regression testing process. Using the actual GUI has over time shown to be a complicated task and is therefore often broken out as its own standalone test object. This study looked into time and quality constrains of using the GUI as driver of the regression testing of business requirements in a web based solution. By evaluating the differences in execution times of test cases between Application Programming Interface (API) calls and GUI driven testing, flakiness of test results and required modifications over time for a specific test suite. These constraints were analyzed by looking into how reliability of the test results could be achieved. With a GUI driven full end to end scope the quality in software solutions could be improved with a reduction in the number of interface issues and detected errors in deployed systems. It would also reduce the volume of test cases that needs to be executed and maintained as there are no longer standalone parts to verify separately with partially overlapping test cases. The implementation utilized Selenium WebDriver to drive the GUI and the results showed that by utilizing Selenium the test execution times were increased from approximately 2 seconds (API) to 20-75 seconds (Selenium). The flaky test results could be eliminated by applying the appropriate pattern to detect, locate, and scroll into visibility prior to interacting with the elements. In the end of the study the test execution results were 100% reliable. The navigation required 15 modifications over time to keep them running. By applying the appropriate pattern a reliable test result can be achieved in end to end regression testing where the test case is driven from the GUI, however with an increase in execution time.

Keywords: Selenium, WebDriver, Regression testing, GUI driven testing, Single Page Application, SPA, Application Programming Interface, API

Preface

This thesis would not have been possible without the help of all marvelous colleagues at Svenska Spel. I’m grateful for all input, comments and to all those who acted as soundboards during the time I spent implementing FIA and writing this thesis.

A special thanks to Michael Olofsson and Jonas Wadsten at Svenska Spel for letting me share your test resources during testing of Vikinglotto.

To the ladies in my life Arja, Emilia, Hilma and Fia, you are my inspiration and the ones that make everyday a Joy to live, you are all dear to me and I cannot express enough how much I love you all.

Contents

1 Introduction

In July 1969 a man stepped on the moon for the first time. Almost 50 years later with all the technology advances available, we still struggle to run regression testing of interactive web applications in a successful and cost efficient way.

The repetitive task of regression testing can be one of the costliest testing activities performed in the software development project. The same test cases are repeatedly executed in order to ensure that introduced code changes do not break anything in the application. Due to the frequency and repetitive nature of the task the execution depends heavily on automation of test execution in order to achieve both speed and cost efficiency in the regression test activity.

Automated testing of single-page applications (SPA) [1], [2] is hard to achieve as execution of test cases takes long time, regularly breaks and can result in flaky test results. Due to these issues testing is often split in different parts, which are tested standalone from each other.

This thesis studies the problems observed during regression testing of a commercially available web application and proposes solutions on how to overcome these issues, with the intention to avoid splitting up the testing of the different components into separate objects.

1.1 Background

An SPA is logically divided into two different entities (figure 1.1), the frontend and the backend. SPAs implemented today are implementing the same kind of functionalities as ordinary desktop applications are, and could be seen as ordinary desktop applications.

The frontend implements the graphical user interface, used by the user to interact with the system. In an SPA the GUI is realized as a web page running in an ordinary web browser. The communication between the frontend and backend is done over an application programming interface (API). Which enables the frontend to fetch and send data from/to the backend.

The backend implements the business logic needed by the system such as for example file/database access to retrieve data needed for the realization of the functions.

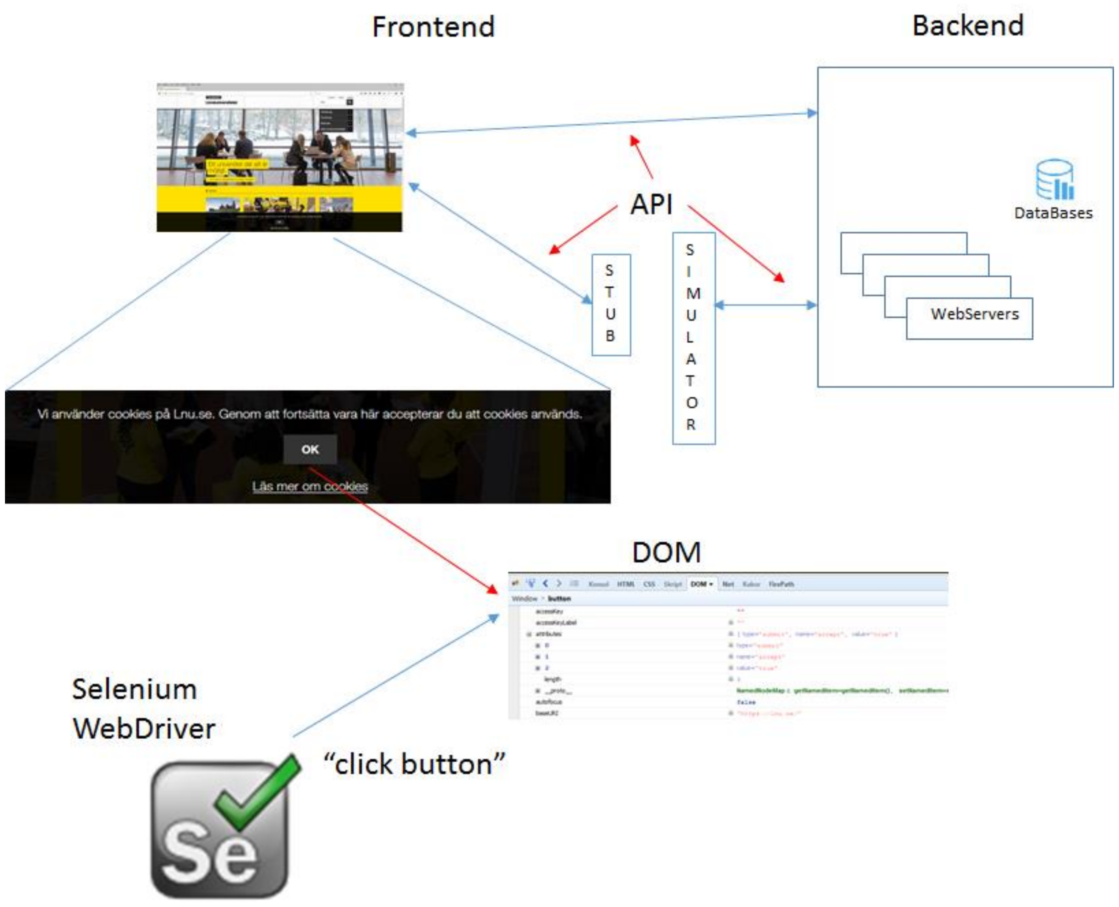


Figure 1.1 Test Automation of SPA

1.1.1 Regression testing

Testing is done in order to build confidence that the software works as intended. When a piece of code is modified, regression test is performed and as described in the IEEE definition a selected subset of test cases is executed.

Regression test does not test new or modified functionalities. Regression testing [3] focuses on making sure that the changed code did not break any existing functionality.

Development of a software application is an iterative process, causing numerous software changes of already tested and approved code. These iterations cause a need for retesting of the code, regression testing. Due to the repetitive nature of the regression test activity it can be one of the most expensive test activities, indicated by Chittimalli et al. [4] 80% or more of the testing budget is consumed by regression testing. Further 50% of the total budget for software maintenance is spent on testing.

The IEEE definition of regression testing [5] is *Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.*

1.1.2 Interactive web applications

An interactive web application is in most cases logically separated in two parts. A backend and a frontend. The logical separation is needed and used for security reasons.

1.1.3 Backend

The backend system in a web application implements the business logic. It also provides the application with the needed access to persistent data storage, often implemented using some kind of database.

1.1.4 Frontend

The frontend implements the GUI needed for the user to interact with the web application. A web browser is ordinarily used for the access to the application. However the web browser itself does not implement the logic of the GUI. What it does is providing the infrastructure needed for a single-page application (SPA) to run in.

Because of security reasons the frontend does not have direct access to persistent data storage.

1.1.5 API

An application programming interface (API) is used for sending commands and data between the frontend and backend.

1.1.6 STUB and Simulator

A STUB and/or Simulator is used when testing the components separately from each other. They act as the “other” entity and responds to API requests in a predefined manner.

1.1.7 Document Object Model

The web browser keeps track of the web elements implemented by the SPA in an internal register, document object model (DOM) [6], [7]. Different components such as buttons, text fields etc. which are available in the GUI, the locations and states of them are as well stored in the DOM.

1.1.8 Selenium WebDriver

Selenium WebDriver [8] is an open source application that provides a programmable interface to interact with the web browser. The user can control various web elements over this interface, such as clicking buttons, entering text in text fields or validate strings of text etc. Thus automatically performing the tasks a user would do while using the web application.

The web browser stores the available web components in an internal register (DOM). It is the DOM that Selenium WebDriver interacts with to perform the requested tasks.

1.1.9 The regression test activity of an SPA

Regression testing of an SPA is often divided into 3 different activities.

1. Testing of the frontend (GUI) using various techniques, where both the look and feel of the application is tested, as well as how to navigate, the usability and clarity of the information presented.
2. Test of the backend (business logic) is normally done by triggering the API requests which the GUI is using. However as this is done without using the GUI, the problems with the navigation and slowness of the application running in the browser are avoided.
3. A limited test suite consisting of some test cases that tests that the two entities (frontend and backend) can interact with each other and some scenarios are tested.

1.2 Previous research

As indicated by Zarrad A. [9] regression testing as such is a research area that is covered. The main focus has been on more traditional testing techniques such as test cases select and test case reductions, regression testing of both GUIs and Databases. The Systematic review performed by Zarrad however indicates that regression testing of web based applications are not commonly researched.

Automation of regression testing can be conducted using tools such as Jmeter [10] or other Capture and Replay tools as indicated by Leotta M., et al. [11], [12] or by applying tools which are programmatically in control of the browser driving the tests [13], [14]. The difference is both in how the test case is designed as well as how it is executed. In a Capture and Replay scenario the interaction with the web browser is recorded while performing a specific task. The recording is then later on replayed during testing. This however makes the test case vulnerable for changes which would break the test execution as indicated by Leotta M., et al [14]. Using a Capture and Replay approach regularly broke the test execution and re recordings of test cases on a regular basis were required [12], [15].

Other research in the area of GUI or GUI driven testing has focused on the page object pattern. The patter is utilized in order to limit the need for maintenance of test cases [12], [15], where objects are logically grouped instead of treated as single objects. Typically the login box is considered as a page object which requires filling in a username, a password and clicking the login button.

As there is a lot of research already done, *so what is new with this research?* Previous research has focused on methods to limit the scope of the testing in various ways. The problematic GUI has also been removed from the equation due to its slowness and problems in keeping test cases running, which has increased the total volume of test cases needed for regression testing. The new research that this report includes is to investigate the problems from a new approach, instead of reducing the scope and removing components from the equation, it studies the issues preventing the GUI from being used in testing. The reasons that prevent the possibility to combine test cases using the GUI are studied by:

1. Studying the problems that arise during the execution of the test cases
2. The study is conducted on a commercially available web application, which is not always the case when testing of web solutions are researched.
3. Measures to address issues that arise are also tested and evaluated during the study

1.3 Problem formulation

The activity of regression testing is retesting that nothing has broken in the existing functionalities of the modified application. However when testing of an SPA is split into separate standalone activities the interoperability between frontend and backend components is not tested, or is only partly tested.

This introduces a risk that interoperability issues are never found during the regression test activity. The volume of test cases also increases, as several partly overlapping test cases are needed to cover each part as standalone components.

The navigation performed in the GUI requires its own set of test cases with a simulated backend. The business logic provided in the backend requires its own set of test cases where the frontend is simulated. This results in partly overlapping test cases. The time that might be gained by regression testing the components separately comes with a cost in form of additional test cases and a risk that interoperability failures are never discovered during testing of the SPA.

By studying and addressing the factors limiting the use of the GUI in regression testing, the intention is that the total amount of needed test cases can be reduced if the functionalities are tested end to end using the GUI. The test coverage is also improved and fewer test cases needs to be maintained.

1.4 Motivation

Upto 80% of the testing budget can be spent on regression testing. As the usage of the real GUI in driving the testing is difficult it is vital to understand the factors causing these difficulties. By addressing them with proper measures the need to divide the testing activity into separate entities (frontend and backend) decreases. As interoperability is also tested the test coverage is increased and partly overlapping test cases can be reduced.

1.5 Research Question

RQ1.	What are the top 2 issues using Selenium WebDriver that cause longer test case execution times when tests are run using a browser, compared to testing the same functionality with APIs, and what can be done to minimize the time differences?
RQ2.	What are the top 2 issues using Selenium WebDriver causing flaky (unreliable) test results, and how can these be overcome?
RQ3.	What are the top 2 issues using Selenium WebDriver causing test cases to break when GUI modifications are done, and how can these be overcome?

The expected outcome of the project is knowledge about the main issues preventing the web GUI from being used as the driver in regression testing of the business requirements in a web based application, which prevents the whole application to be tested together.

The project also expects to propose measures to address the issues found and show the difference in the results when those measures are applied.

1.6 Scope/Limitation

There are many different test solutions available providing different kinds of functionalities. This project limits the use of test tools to Selenium WebDriver for communication with the web browser.

This report does not compare different web browser driving tools. The aim of this research is to understand the underlying factors that prevents testing of frontend and backend together.

It is assumed that a lot of different factors come in play, however the project focuses on the factors that are related to the execution time of the test case, robustness and reliability of the test result.

There are a lot of different SPAs available on the market and this project uses one of them, Svenska Spels' implementation of a few of their betting products. Some of the findings made might not be applicable to other SPAs, as this depends entirely on the nature of the issues that are found.

The test execution is expected to be slower when tests are run via a web browser compared to using direct API calls for testing of the business logic. However a decent good enough execution time is expected to be achieved.

The SPA uses JavaScripts to implement its functionality. This creates a dependency on the used web browser, as they implement JavaScript in different ways, rendering different performances. The experimentation is limited to using Mozilla Firefox 48.0-53.x and Google Chrome 58.x.

1.7 Target group

The target group is Software Testers of web applications that faces similar issues and any test organization that wishes to enhance the confidence of their regression testing.

1.8 Outline

The following chapters outline how the experimentation was done, the result and final outcome of the experiment.

Firstly the method is described to give an overview of how the project was conducted, and which data was collected to address the stated questions.

Secondly the implementation of the FIA application is described, followed by a presentation of the collected data.

In the final part of the thesis the data is analyzed, discussed and summarized in a conclusion. The final chapter also addresses issues were further research would make sense.

2 Method

The stated research area contained aspects that were addressed using both a quantitative research method, and areas where a qualitative research approach was used.

2.1 Scientific Approach

It was assumed from the beginning that the problems would not be visible immediately and the study needed to be conducted over a longer time period. For this reason a quantitative research method was selected for data collection.

The quantitative research was used to find the areas where most of the problems aroused. It was primarily used for understanding of *when* the problem occurred, and *what* the problems were that occurred.

A quantitative research method would however not provide additional information on *why* and *how* a problem occurred. For this reason qualitative research was used to investigate *why* and *how* the problem occurred.

A combination of both methods were then used in order to understand the full scope of *what*, *when*, *why* and *how*.

2.2 Method Description

The research addressed three topics. Firstly the execution time required for the test execution. Secondly the correctness and reliability of the test results. Finally the area of ensuring that testing could continue even though changes were introduced.

The independent variable was the set of used test cases (tables 4.1-4.3), while the dependent variables were the ones affected by the test run, namely execution time, test result reliability and the identified reasons that caused a test case to fail when it worked on a previous release of the software.

Better reliability in the figures measured was achieved by executing the same test suite for 100 times. As RQ2 and RQ3 would give unreliable results if they were studied only for a short period of time, the study was conducted for at least 20 weeks. 20 weeks was the time span used for the implementation of a new product, hence this study covered the full cycle of the development project.

2.2.1 Required execution time

The collection of the execution time spent to perform a task was a quantitative task. The execution time required for each API call was measured during the test execution. As the testing of a single business requirement involved several different API calls, the total time spent for testing a business requirement was also collected. This resulted in both an overall picture of the total execution time, as well as a detailed picture of how much execution time each API call consumed. The description of the API calls required to place the bet is seen in figure 2.1.

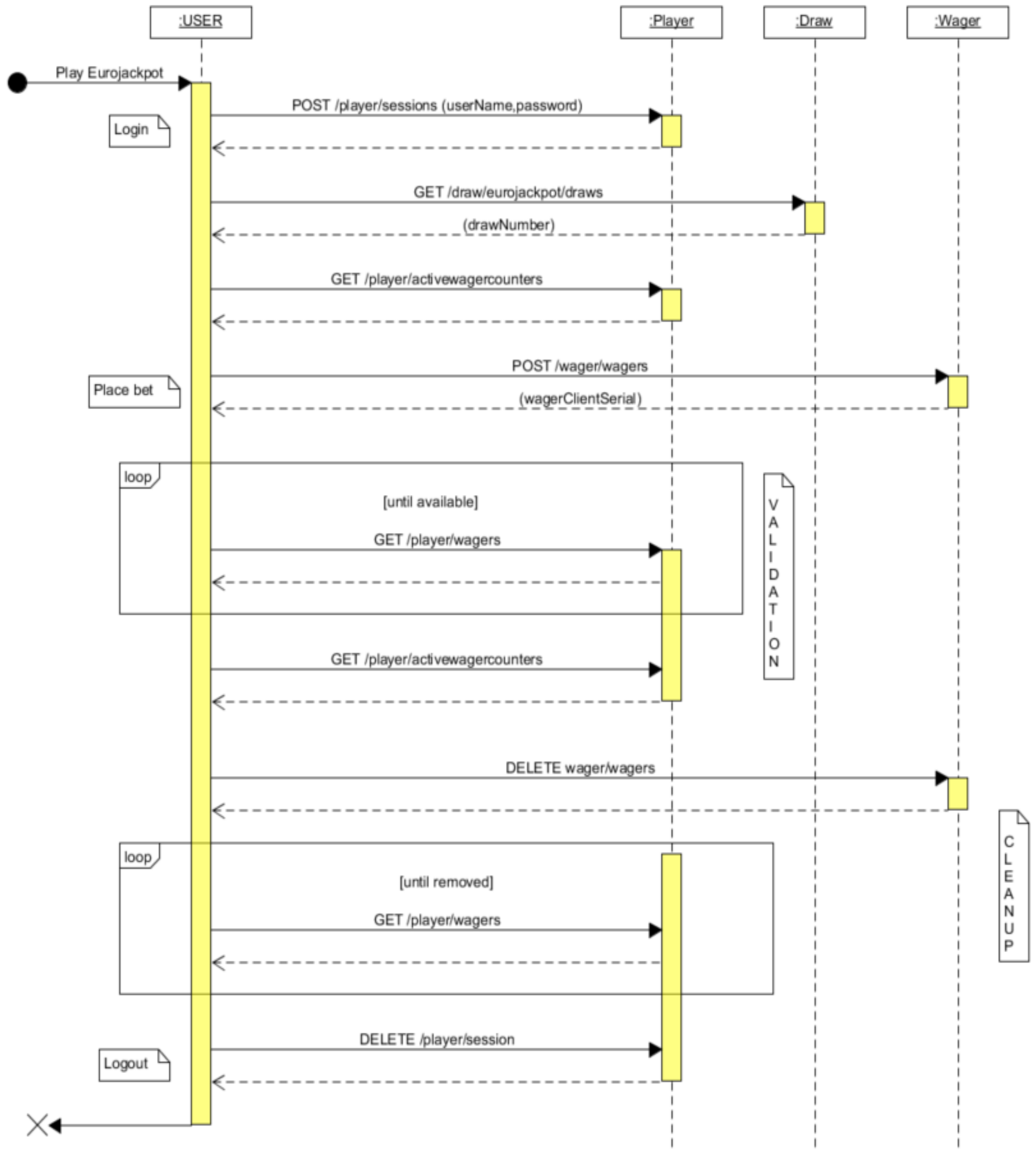


Figure 2.1 API Flow Eurojackpot

The same test case was then executed using a GUI driven approach. In the GUI driven approach the execution times for each API calls could not be measured in the same way, as it was unknown what APIs the SPA actually would use. Instead the execution time of each task was measured, as well as the total required execution time. The tasks required to place the same bet is outlined in figure 2.2.

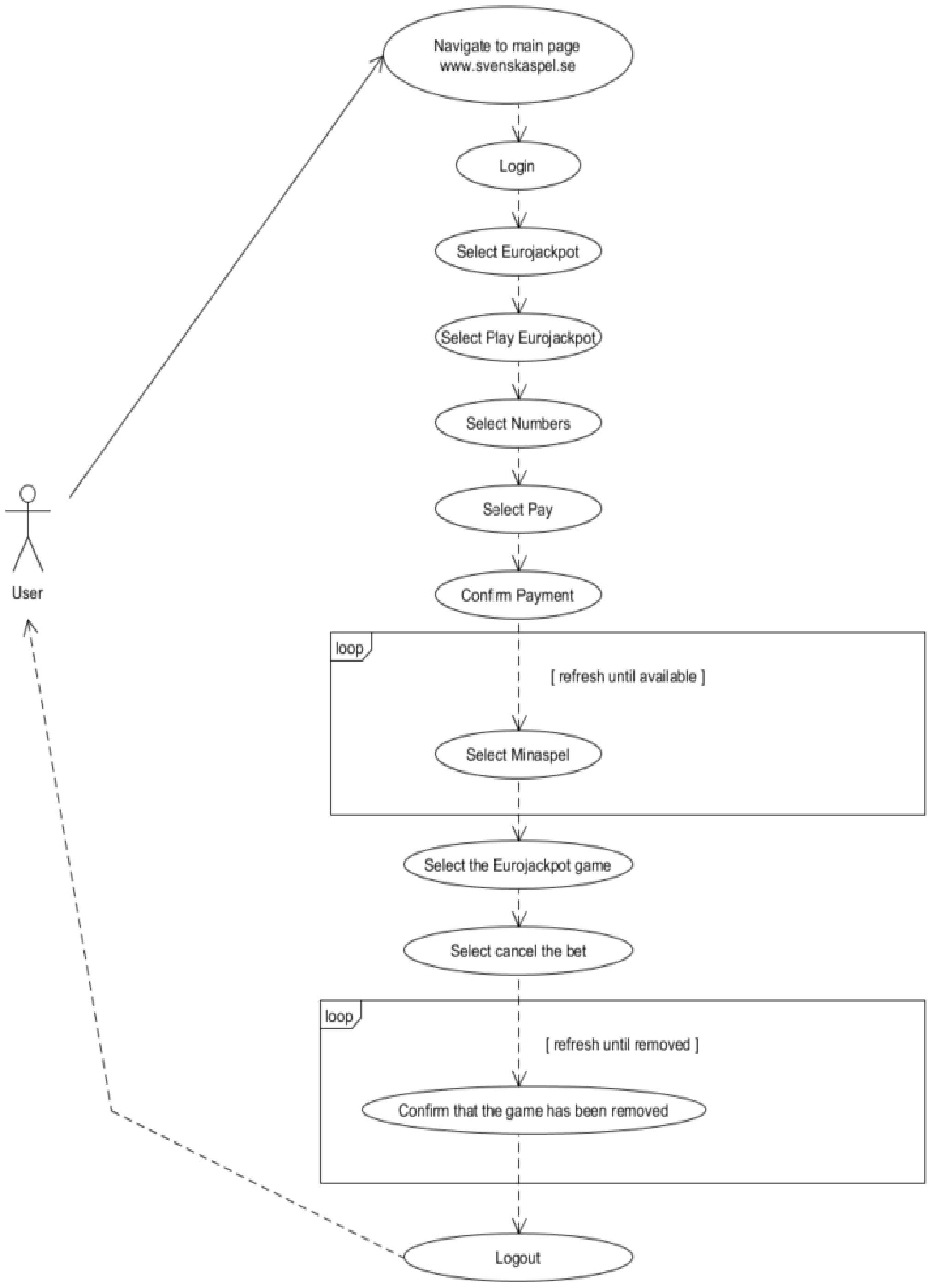


Figure 2.2 GUI Flow Eurojackpot

The results of the API tests were compared with the results of the GUI driven testing, and the areas where the test execution times deviated the most was further analyzed.

For comparable reasons the tasks required to place the bet were grouped into 7 steps (table 2.3). This was required in order to be able to compare a single API call for example POST wager/wagers, which was used to place a bet, with the navigation of filling in all the marks, selection of payment, ordering payment and finally confirming the payment. These were the procedures in the GUI to create the input for the browsers equivalent POST wager/wagers request.

Step	Description	Comment
1	Start a web browser	Not applicable when API calls are used
2	User Login	
3	Initialize the game	
4	Place the bet	
5	Validate placed bet	
6	Remove bet and validate that bet has been removed	
7	User Logout	

Table 2.3 Scenario steps

It was not possible to say in advance what kind of measures that might be needed in order to address the difference in execution times. However the following step was to modify the GUI navigation to optimize and improve the execution times. The modifications needed to be evaluated by retesting the same test case again.

This was an iterative task which was conducted several times before a good enough execution time was achieved.

The delta in the implementation between the first and the last test execution was considered as the required measures in order to improve the execution times. The execution times, addressed areas and the modified delta were considered as input to the answer to research question RQ1.

2.2.2 Reliability of test results

Tests driven by the GUI often gave flaky results. Sometimes they worked and sometimes they did not. To be able to trust the received test result flakiness could not be tolerated.

Each of the used test sequences was first tested manually to ensure that the implemented business requirement was working. The used test suite was then executed several times, with the expectation of a 100% success rate.

During the test execution any failure in the results was registered, as well as which task that failed. Each failure was then analyzed further to understand the reason behind the failure. Once the reason was known modifications to the navigation were applied and the test case was retested, using the same approach as required for understanding of the execution times.

The addressed areas and the modified delta were considered as answer to research question RQ2.

2.2.3 Modifications of the GUI

Using the GUI to drive the testing introduced a vulnerability for breaking the test cases when new components were introduced, or when old components were removed or replaced.

The third research question addressed the issue of keeping the test cases successfully running by limiting the effects caused by the vulnerability. This question was addressed when the previous two research questions had been answered. At that point it was expected that the test suit was executed fast enough and that the result was reliable. The third question was answered by analyzing how often the test execution failed and where the solution required modifications to the navigation of the test execution.

RQ3 was answered by counting the number of occasions when the navigation failed due to moved or replaced elements which caused the sequence of placing a bet to fail.

It was not only the SPA that was continuously being changed, also the software components around the SPA were being updated. Both the web browser and Selenium, as well as used drivers were being changed and improved. Old bugs were fixed, and new ones were introduced. The aim was always to use the latest software version of each test tool. This required a continuous retesting and reevaluation during the entire project.

The upkeep of software versions was not considered as input to RQ3.

2.3 Reliability and Validity

The software components used for the GUI navigation were continuously changed. Hence limitations and restrictions in the current implementation affected the areas were most problems aroused differently, depending on which versions of the software (SW) components that were used.

The experimentation was performed in four different test systems. While the test systems provided similar functionalities, there were differences in the capabilities of the hardware (HW). Execution times of a test suite would vary depending on which test system that was used. All measurements related to execution times were in the end collected from the same test system. This ensured comparability of the measured values.

In some occasions the test system was shared with other users, which might have had a negative effect on the outcome of the test execution.

The SPA was using JavaScript to render its required functionalities. This created a dependency to the used web browser, as they implement JavaScript in different ways. Firefox versions 48-53 and Google Chrome 58.x were used during the experimentation.

2.4 Ethical Considerations

The research was conducted in the form of experimenting on an existing product, no ethical considerations were foreseen.

3 Implementation

In the project a number of different deliverables were implemented.

Primarily a set of test cases for testing of the business requirement related to placing the bet was implemented.

Secondly a test loader was implemented to execute the suite of test cases. The reason why a loader was implemented was to get full control of the execution without having to overcome with various tool related features or characteristics. Ordinarily a test tool comes with a set of generic components that generates a certain overhead in execution times, as well as needed workarounds to handle specific requirements or to cope with different features of the tool. These were unwanted options that were removed by making an own implementation of a test loader FIA (figure 3.1).

FIA is a platform independent JAVA implementation.

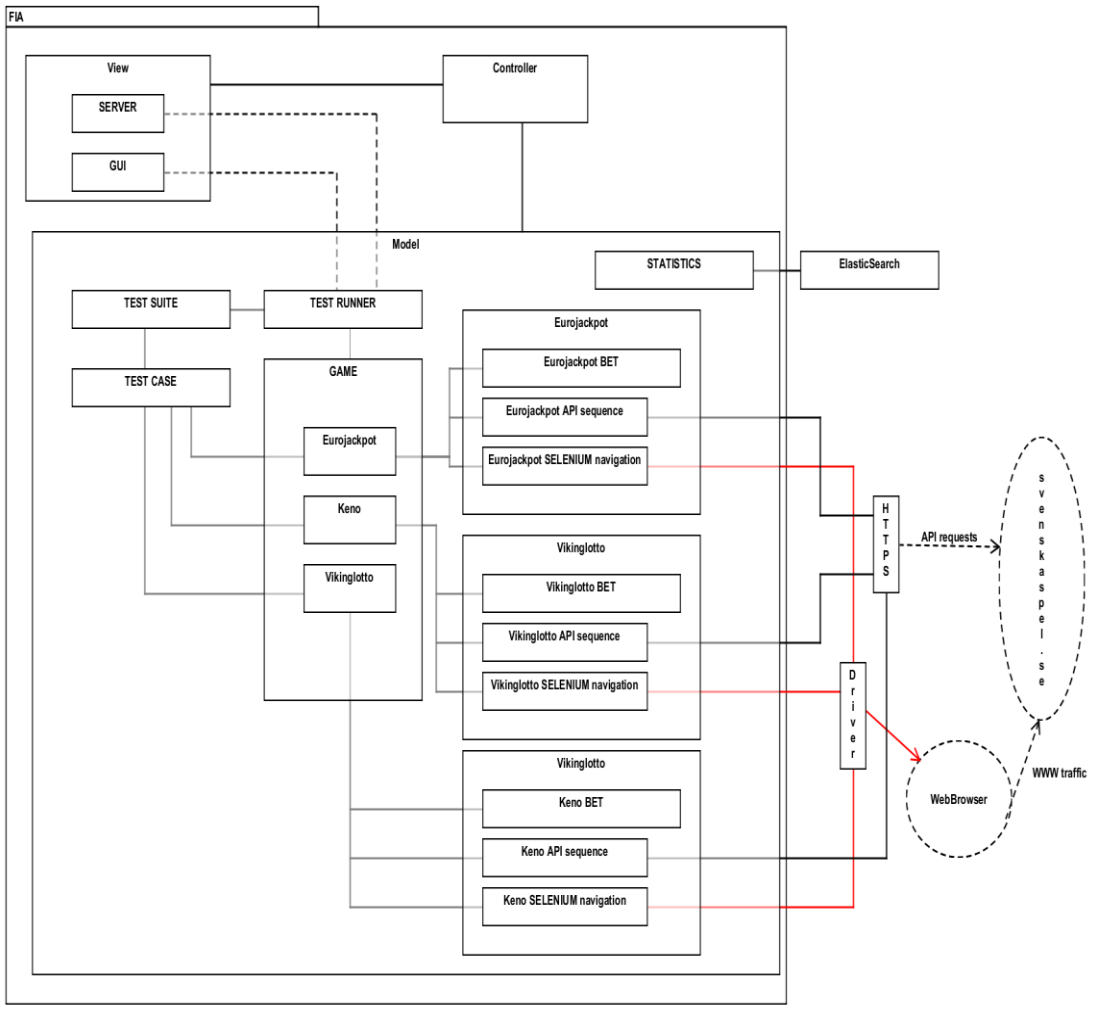


Figure 3.1 FIA overview

3.1 Components of FIA

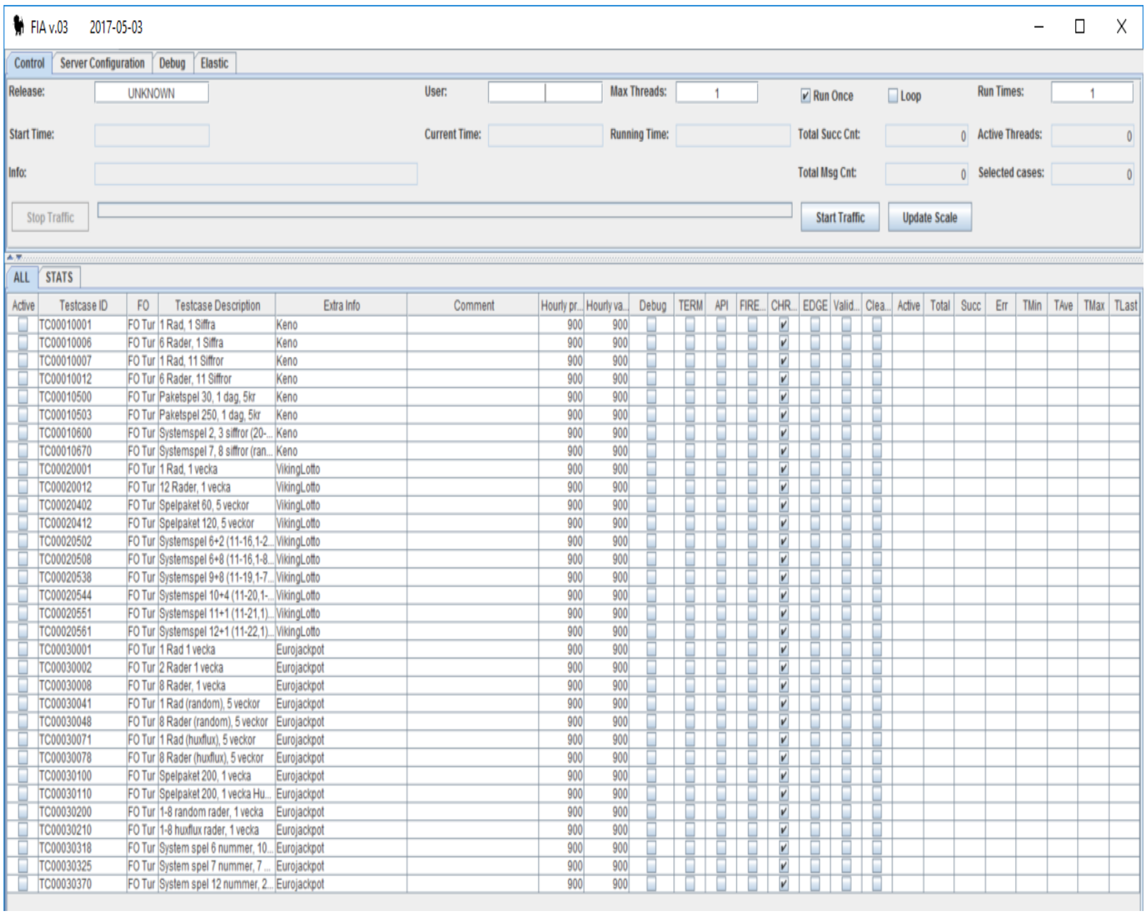
The components of Fia were divided into a Model-View-Controller (MVC) pattern.

3.1.1 View

FIA runs in two different modes, either with a GUI, shown in figure 3.2, or in a server mode. The implementation of FIA is generic, and hence any web browser could be used for the execution, and it would run on any operating system supporting JAVA.

However Edge and Internet Explorer (IE) are only available on the Windows operating system. Due to this there is only a very limited need for a pure server solution, as the browser depends on the desktop for the execution. The server mode is however used when running the API test cases, as those are text base, and do not depend on the desktop to execute.

The internal GUI (figure 3.2) of the tool is primarily for monitoring and follow up of the progress of the test execution.



Release: UNKNOWN		User:	Max Threads: 1	☑ Run Once ☐ Loop	Run Times: 1
Start Time:		Current Time:	Running Time:	Total Succ Cnt: 0	Active Threads: 0
Info:				Total Msg Cnt: 0	Selected cases: 0

Active	Testcase ID	FO	Testcase Description	Extra Info	Comment	Hourly pr...	Hourly va...	Debug	TERM	API	FIRE...	CHR...	EDGE	Valid...	Clea...	Active	Total	Succ	Err	TMin	TAve	TMax	TLast
☐	TC00010001	FO Tur	1 Rad, 1 Siffra	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010006	FO Tur	6 Rader, 1 Siffra	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010007	FO Tur	1 Rad, 11 Siffror	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010012	FO Tur	6 Rader, 11 Siffror	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010500	FO Tur	Paketspel 30, 1 dag, 5kr	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010503	FO Tur	Paketspel 250, 1 dag, 5kr	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010600	FO Tur	Systemspel 2, 3 siffror (20-...	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00010670	FO Tur	Systemspel 7, 8 siffror (ran...	Keno		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020001	FO Tur	1 Rad, 1 vecka	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020012	FO Tur	12 Rader, 1 vecka	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020402	FO Tur	Spelpaket 60, 5 veckor	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020412	FO Tur	Spelpaket 120, 5 veckor	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020502	FO Tur	Systemspel 6+2 (11-16,1-2...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020508	FO Tur	Systemspel 6+8 (11-16,1-8...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020538	FO Tur	Systemspel 9+8 (11-19,1-7...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020544	FO Tur	Systemspel 10+4 (11-20,1-...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020551	FO Tur	Systemspel 11+1 (11-21,1)...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00020561	FO Tur	Systemspel 12+1 (11-22,1)...	VikingLotto		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030001	FO Tur	1 Rad 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030002	FO Tur	2 Rader 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030008	FO Tur	8 Rader, 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030041	FO Tur	1 Rad (random), 5 veckor	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030048	FO Tur	8 Rader (random), 5 veckor	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030071	FO Tur	1 Rad (huxflux), 5 veckor	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030078	FO Tur	8 Rader (huxflux), 5 veckor	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030100	FO Tur	Spelpaket 200, 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030110	FO Tur	Spelpaket 200, 1 vecka Hu...	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030200	FO Tur	1-8 random rader, 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030210	FO Tur	1-8 huxflux rader, 1 vecka	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030318	FO Tur	System spel 6 nummer, 10...	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030325	FO Tur	System spel 7 nummer, 7 ...	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								
☐	TC00030370	FO Tur	System spel 12 nummer, 2...	Eurojackpot		900	900	☐	☐	☐	☐	☑	☐	☐	☐								

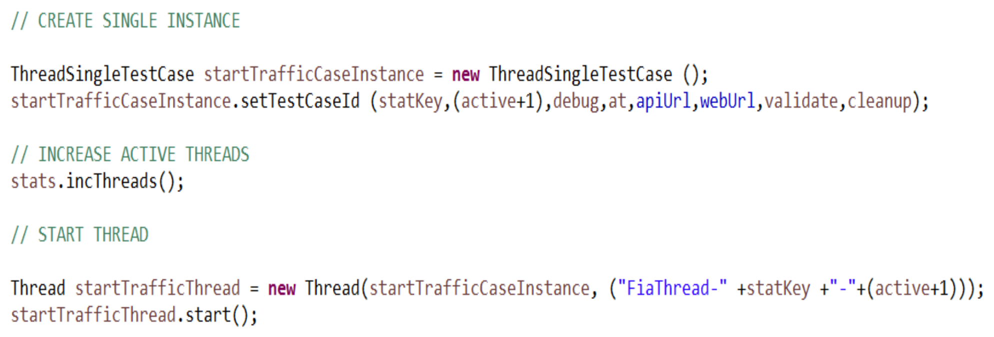
Figure 3.2 GUI of FIA

3.1.2 Model

The model is the Core implementation of FIA. It consists of a TestRunner, TestSuite, TestCases, supported Games and a Statistics component.

TestRunner

The TestRunner is a multi-threaded implementation. Upon start of execution of the test case, a new thread is created and the execution of the test case is contained in that thread. Upon completion of the test case the thread is killed. As each test case is running independently a failing test case does not affect any of the other test cases (code 3.3).



```
// CREATE SINGLE INSTANCE

ThreadSingleTestCase startTrafficCaseInstance = new ThreadSingleTestCase ();
startTrafficCaseInstance.setTestCaseId (statKey,(active+1),debug,at,apiUrl,webUrl,validate,cleanup);

// INCREASE ACTIVE THREADS
stats.incThreads();

// START THREAD

Thread startTrafficThread = new Thread(startTrafficCaseInstance, ("FiaThread-" +statKey +"-"+(active+1)));
startTrafficThread.start();
```

Code 3.3 Execution of Threads

The same is applicable if multiple instances of the same test case should be executed. At start of each case a unique thread for each test case is also started.

This approach was selected to avoid the loader becoming an issue related to RQ1-3. The loader should not cause the test cases to fail. Failure should be related to either the externally used SW components, the navigation or that testing of the business requirement(s) fails.

The test suite can be executed in different ways by the loader, where the following options are possible:

1. Case by Case.
   One test case at a time is executed. The following test case is started upon completion of the previous one.
2. Maximum number of Threads.
   The test cases run in parallel mode up to the maximum number of allowed concurrent test cases.
3. Hourly Traffic profile.
   A required amount of test threads are started per hour. The configuration is set per test case, not on the full test suite and the load is evenly spread out during a one (1) hour time period.

In the first two test modes it is also possible to indicate how many times the entire test suite should be looped, or if it should run until manually stopped.

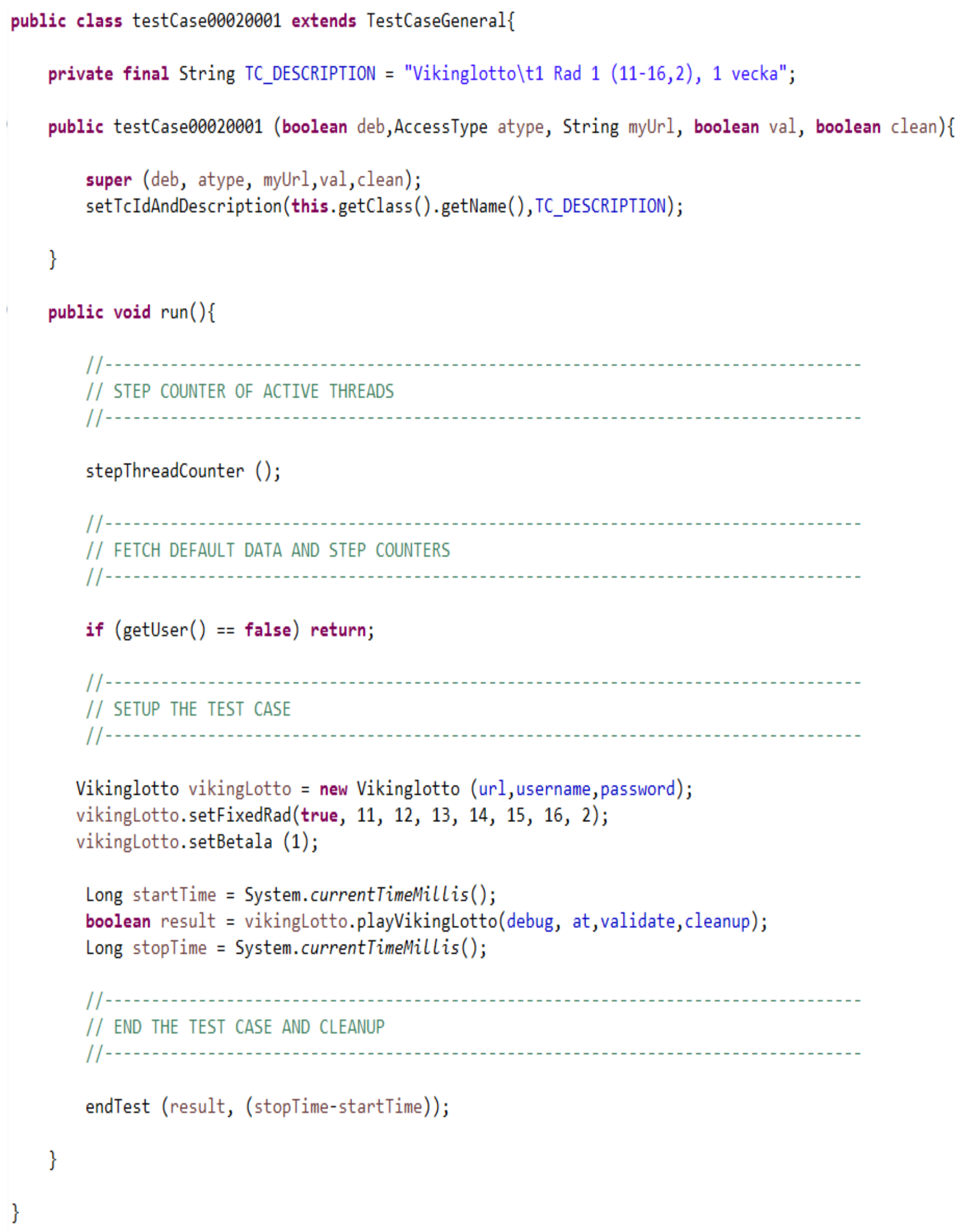
TestSuite
The test suit is a csv formatted list of required test cases, with a set of parameters:

- The test case(s)
- The requested traffic level
- Interface to be used
- Hourly profile

The interface to use indicates that the case should either be executed via the API or via any of the 3 currently implemented browsers (Firefox, Chrome and Internet Explorer).

TestCases
Each test case is implemented in its own class file, by extending the general test case class. The General class contains common methods such as setting ID of the test case, a common start up sequence, requesting a player user and a teardown sequence.

The logic of the test case setup in the TestCase class is shown in code 3.4.



```
public class testCase00020001 extends TestCaseGeneral{

    private final String TC_DESCRIPTION = "Vikinglotto\t1 Rad 1 (11-16,2), 1 vecka";

    public testCase00020001 (boolean deb,AccessType atype, String myUrl, boolean val, boolean clean){

        super (deb, atype, myUrl,val,clean);
        setTcIdAndDescription(this.getClass().getName(),TC_DESCRIPTION);

    }

    public void run(){

        //--------------------------------------------------------------------------------
        // STEP COUNTER OF ACTIVE THREADS
        //--------------------------------------------------------------------------------

        stepThreadCounter ();

        //--------------------------------------------------------------------------------
        // FETCH DEFAULT DATA AND STEP COUNTERS
        //--------------------------------------------------------------------------------

        if (getUser() == false) return;

        //--------------------------------------------------------------------------------
        // SETUP THE TEST CASE
        //--------------------------------------------------------------------------------

       Vikinglotto vikingLotto = new Vikinglotto (url,username,password);
       vikingLotto.setFixedRad(true, 11, 12, 13, 14, 15, 16, 2);
       vikingLotto.setBetala (1);

        Long startTime = System.currentTimeMillis();
        boolean result = vikingLotto.playVikingLotto(debug, at,validate,cleanup);
        Long stopTime = System.currentTimeMillis();

        //--------------------------------------------------------------------------------
        // END THE TEST CASE AND CLEANUP
        //--------------------------------------------------------------------------------

        endTest (result, (stopTime-startTime));

    }

}
```

Code 3.4 Test Case

Implementation of the Game components
Each game is implemented in its own set of core components (figure 3.5).

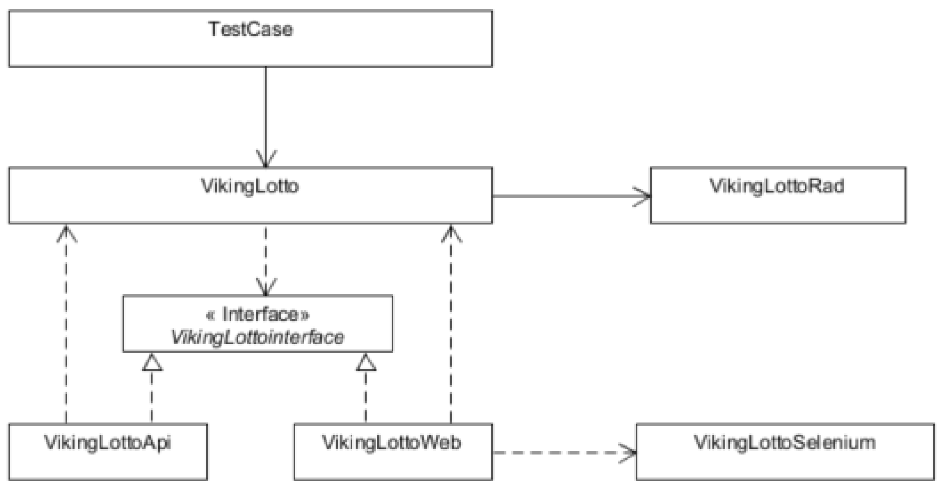


Figure 3.5 Game components

The required configuration on how to setup the bet is done in VikingLotto and VikingLottoRad. The sequence of API requests, or required navigation is implemented in VikingLottoApi respectively VikingLottoWeb, which also uses VikingLottoSelenium as a helper class, implementing the required navigation towards the Selenium package.

Each API message is implemented in its own class (figure 3.6), the reason behind this decision is to make the implementation of each API as independent as possible (code 3.7). Also the data extractor is implemented in its own class for each API message. This results in partly duplicated code, which can be consider as bad practice from an object oriented point of view. However it reduces the risk of breaking other API requests in case something needs to be updated.

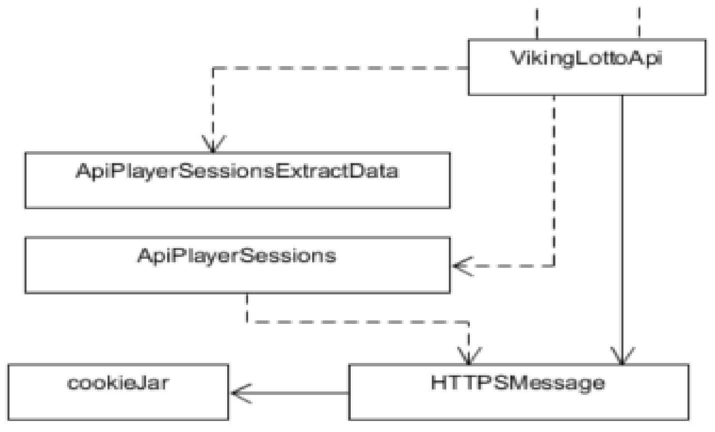
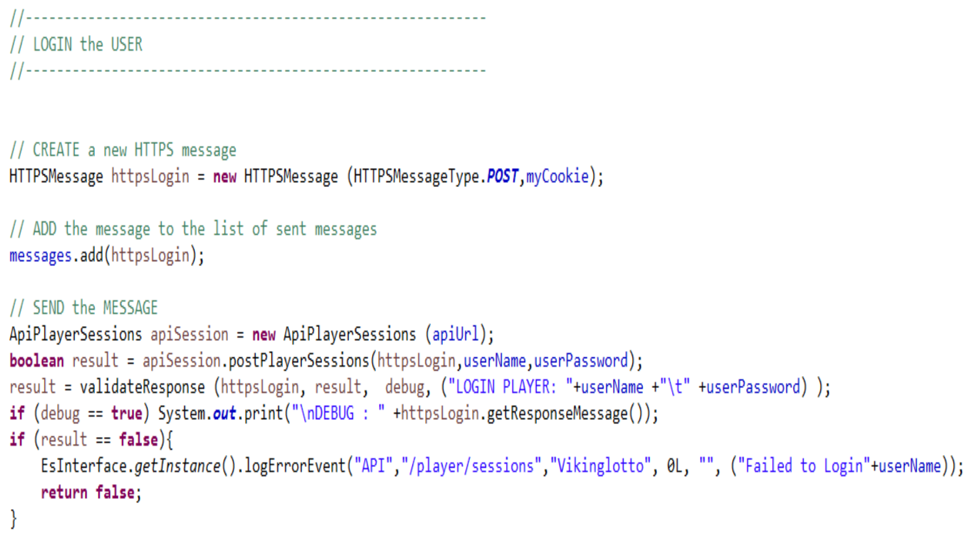


Figure 3.6 API Message sending



```
//------------------------------------------------------------
// LOGIN the USER
//------------------------------------------------------------


// CREATE a new HTTPS message
HTTPSMessage httpsLogin = new HTTPSMessage (HTTPSMessageType.POST,myCookie);

// ADD the message to the list of sent messages
messages.add(httpsLogin);

// SEND the MESSAGE
ApiPlayerSessions apiSession = new ApiPlayerSessions (apiUrl);
boolean result = apiSession.postPlayerSessions(httpsLogin,userName,userPassword);
result = validateResponse (httpsLogin, result,  debug, ("LOGIN PLAYER: "+userName +"\t" +userPassword) );
if (debug == true) System.out.print("\nDEBUG : " +httpsLogin.getResponseMessage());
if (result == false){
    EsInterface.getInstance().logErrorEvent("API","/player/sessions","Vikinglotto", 0L, "", ("Failed to Login"+userName));
    return false;
}
```

Code 3.7 API Message sending example

The web interface uses the selenium WebDriver package to drive the navigation of a web browser. In this example the VikingLottoWeb is the core component for executing the sequence when a bet is to be placed (figure 3.8). VikingLottoSelenium can be seen as a helper class to VikingLottoWeb, it implements the more detailed navigation components of the web page, such as how to click a button, selecting numbers and so on.

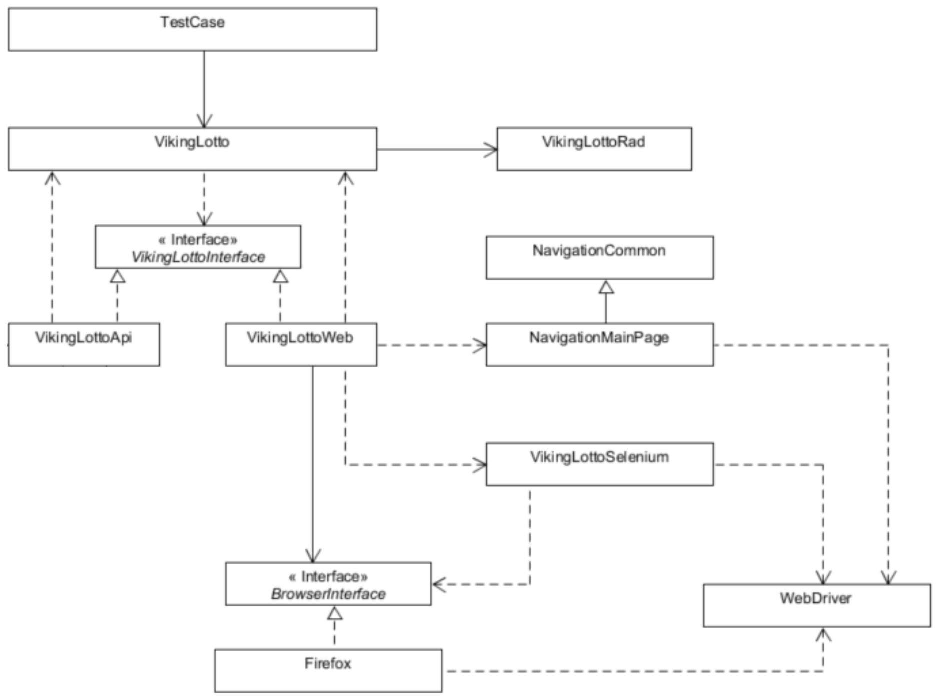


Figure 3.8 Web navigation

This architecture enables easy maintenance of the navigation. By using high cohesion the following is achieved in this architecture:

1. If a new type of web browser is introduced the component affected is mainly the Browser component (Firefox) which is replaced by the unique driver of the new web browser (figure 3.9).
2. The main page navigation is common for all web browsers and all games
3. A new or different game affects only VikingLottoWeb (handing the sequence flow of the web page) and VikingLottoSelenium (implementing the game specific selenium navigation)

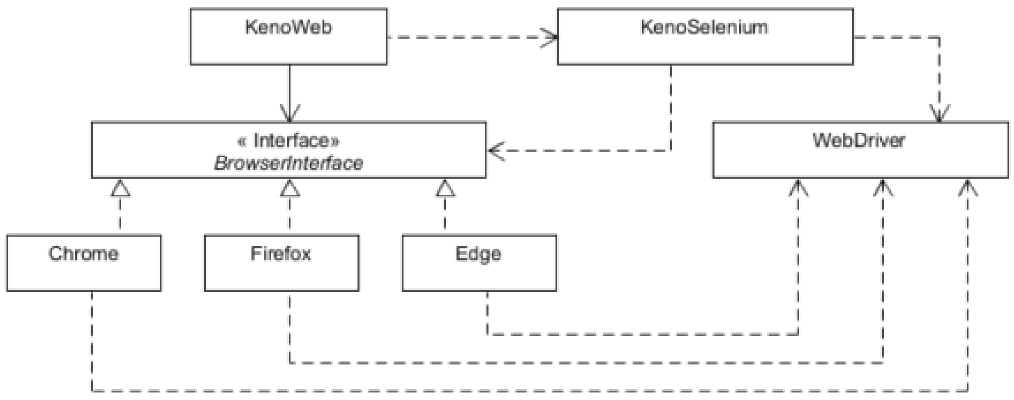


Figure 3.9 Web browsers

The main difference between the different games is the navigation sequence required to place the bet of the game. The implementation of game Keno uses the same structure as game Vikinglotto (figure 3.10). Only the core components that are specific for the game needs to be replaced.

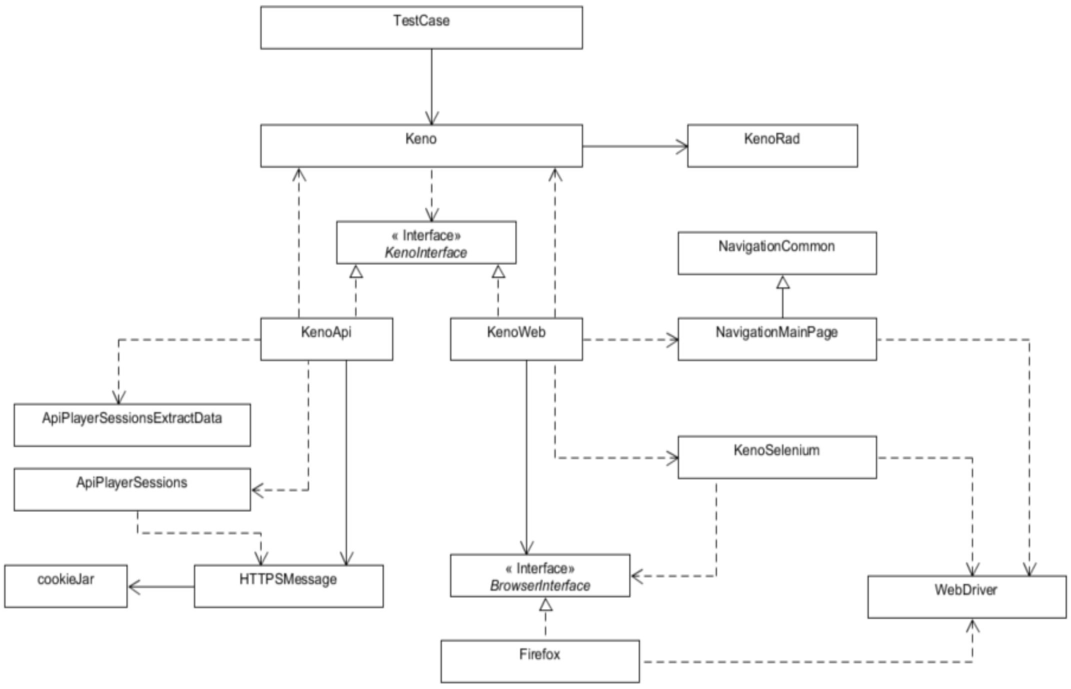


Figure 3.10 Keno Overview

Statistics
Implemented as a singleton providing an interface for storing data in an ElasticSearch database. It also holds the statistical data presented in the internal GUI when the tool is running in the GUI mode.

4 Results

The selected test cases were a subset of the total test suite. The selection consisted of cases that required only a few elements to be selected as well as cases where more elements had to be selected in the GUI. Ranging from the simplest test case with the bare minimum of selectable elements to the worst test case with the maximum of elements to be selected, from a GUI point of view.

Game	**Test Case**	**Description**
Keno	TC00010001	1 Row, 1 Digit
Keno	TC00010006	6 Rows, 1 Digit
Keno	TC00010007	1 Row, 11 Digits
Keno	TC00010012	6 Rows, 11 Digits
Keno	TC00010500	Package game 30
Keno	TC00010503	Package game 250
Keno	TC00010600	System game 2, 3 digits
Keno	TC00010670	System game 7, 8 random digits

Table 4.1: Selected Test cases Keno

Game	**Test Case**	**Description**
Vikinglotto	TC00020001	1 Row
Vikinglotto	TC00020012	12 Rows
Vikinglotto	TC00020402	Package game 60
Vikinglotto	TC00020412	Package game 120
Vikinglotto	TC00020502	System game 6 + 2
Vikinglotto	TC00020508	System game 6 + 8
Vikinglotto	TC00020538	System game 9 + 8
Vikinglotto	TC00020544	System game 10 + 4
Vikinglotto	TC00020551	System game 11 + 1
Vikinglotto	TC00020561	System game 12 + 1

Table 4.2: Selected Test cases Vikinglotto

Game	Test Case	Description
Eurojackpot	TC00030001	1 Row
Eurojackpot	TC00030002	2 Rows
Eurojackpot	TC00030008	8 Rows
Eurojackpot	TC00030041	1 Random row
Eurojackpot	TC00030048	8 Random rows
Eurojackpot	TC00030071	1 Row Huxflux
Eurojackpot	TC00030078	8 Rows Huxflux
Eurojackpot	TC00030100	Package game 200
Eurojackpot	TC00030110	Package game 200 Huxflux
Eurojackpot	TC00030200	1-8 Random rows
Eurojackpot	TC00030210	1-8 Huxflux rows
Eurojackpot	TC00030318	System game 6 + 10
Eurojackpot	TC00030325	System game 7 + 7
Eurojackpot	TC00030370	System game 12 +2

Table 4.3: Selected Test cases Eurojackpot

For a comparable analysis between the usage of direct API calls and GUI navigation using a web browser, the scenarios were divided into the 7 steps, which are listed in table 2.3.

4.1 Test case execution times

Test case	API Execution Time (ms)	Selenium Execution Time (ms)
TC00010001	2082.16	21176.29
TC00010006	2045.30	24230.83
TC00010007	1984.57	22142.45
TC00010012	2102.24	32648.59
TC00010500	2052.89	19344.95
TC00010503	2062.64	19771.95
TC00010600	2137.88	19635.88
TC00010670	2041.09	20317.67
TC00020001	2016.32	21229.98
TC00020012	1998.18	43713.64
TC00020402	2036.84	19212.04
TC00020412	2031.02	19476.82
TC00020502	1977.10	19863.34
TC00020508	2019.29	20362.11
TC00020538	2029.55	20670.02
TC00020544	1994.33	20236.41
TC00020551	2015.50	20320.83
TC00020561	2042.08	20676.85
TC00030001	2036.11	25045.28
TC00030002	1986.22	32009.53
TC00030008	2049.16	73860.56
TC00030041	2033.23	25427.15
TC00030048	2000.31	75222.10
TC00030071	2057.24	21255.97
TC00030078	2100.98	54211.40
TC00030100	2043.81	19355.83
TC00030110	2033.88	19361.66
TC00030200	2018.48	45848.29
TC00030210	2042.44	34839.32
TC00030318	2032.28	20696.83
TC00030325	2016.10	20250.69
TC00030370	2042.21	20657.99

Table 4.4: Average execution times of the test case for 100 test runs

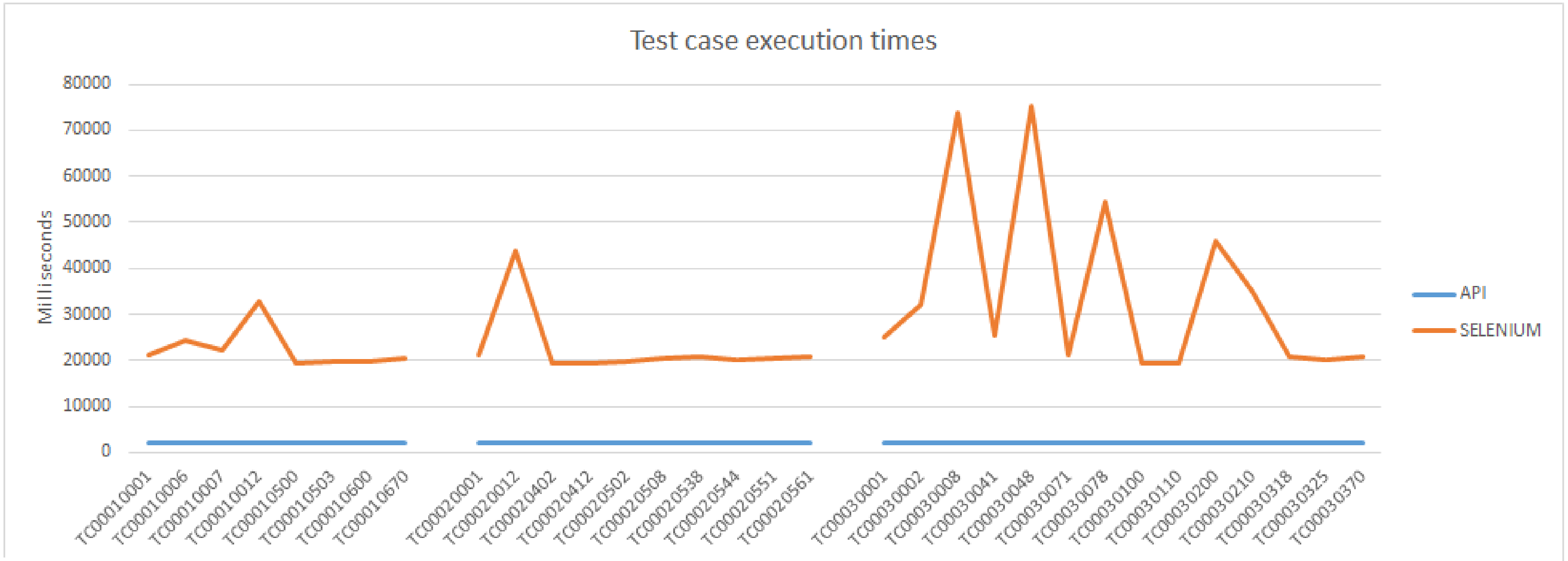


Figure 4.5: Test case execution times for 100 test runs

Game	**Interface**	**Step 1 (ms)**	**Step 2 (ms)**	**Step 3 (ms)**	**Step 4 (ms)**
Keno	Api	*	53.08	97.36	30.67
Keno	Selenium	3378.64	2747.26	1481.44	4820.89
Vikinglotto	Api	*	35.43	116.79	29.15
Vikinglotto	Selenium	3402.94	2768.62	1445.36	4021.34
Eurojackpot	Api	*	38.10	131.14	30.39
Eurojackpot	Selenium	3458.63	2771.23	1380.59	16663.62

Table 4.6: Average execution times for scenario steps 1-4 for 100 test runs
* Not applicable

Game	**Interface**	**Step 5 (ms)**	**Step 6 (ms)**	**Step 7 (ms)**
Keno	Api	973.02	839.05	59.36
Keno	Selenium	1810.65	3479.70	226.34
Vikinglotto	Api	976.39	813.8	34.74
Vikinglotto	Selenium	2517.41	3411.33	231.86
Eurojackpot	Api	1055.87	761.06	40.81
Eurojackpot	Selenium	2327.34	3705.35	230.93

Table 4.7: Average execution times for scenario steps 5-7 for 100 test runs
* Not applicable

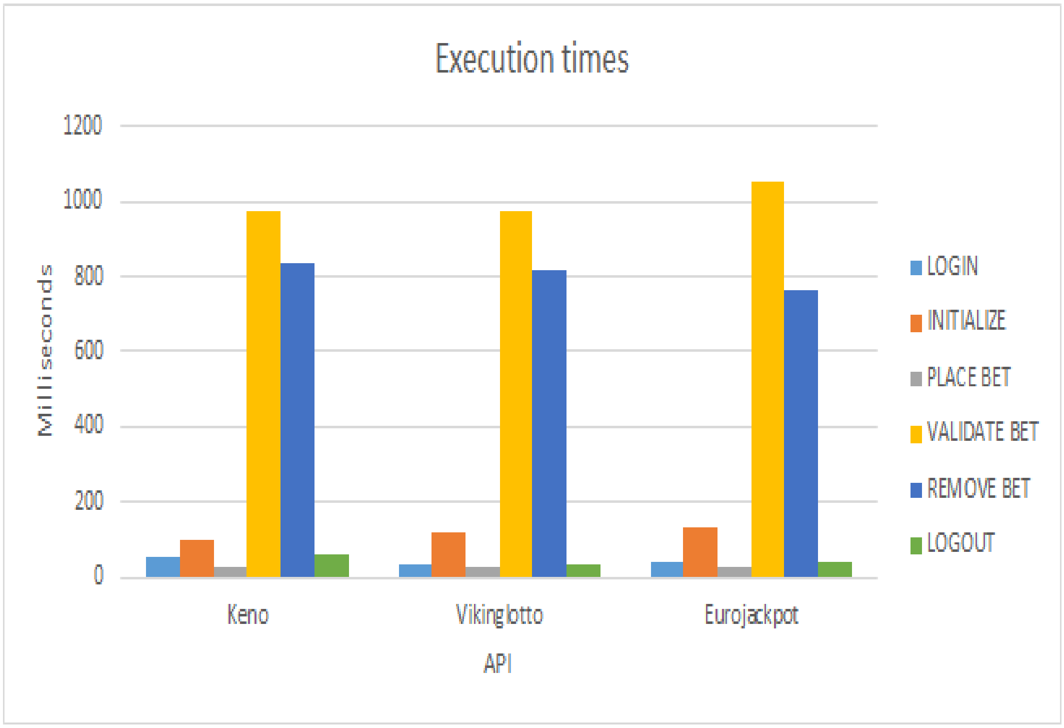


Figure 4.8 Execution times API

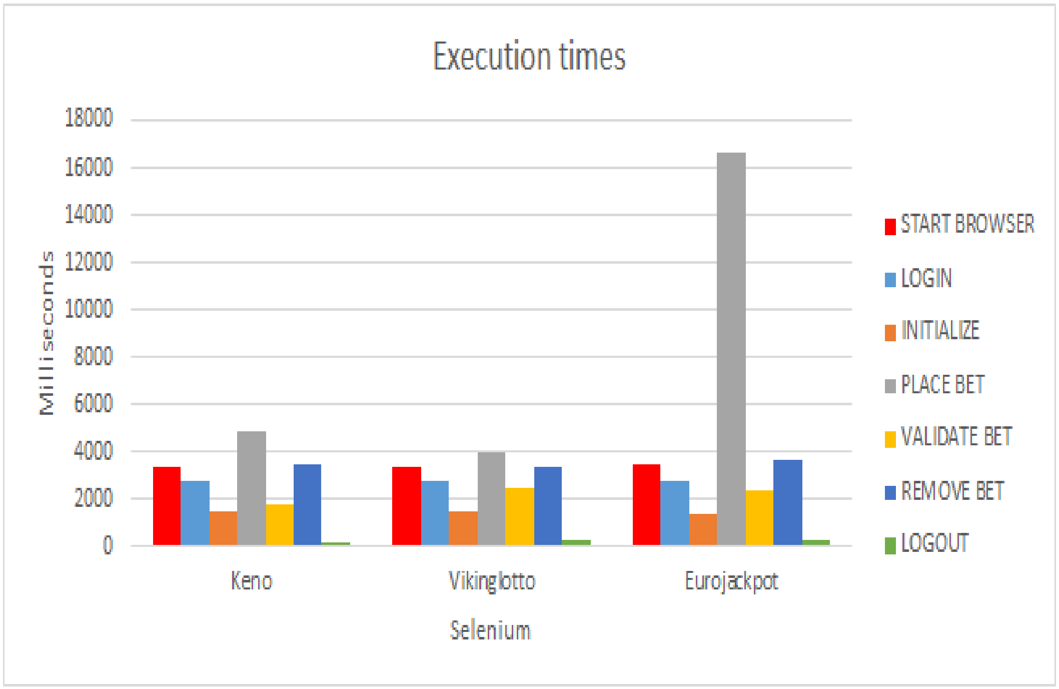


Figure 4.9 Execution times Selenium

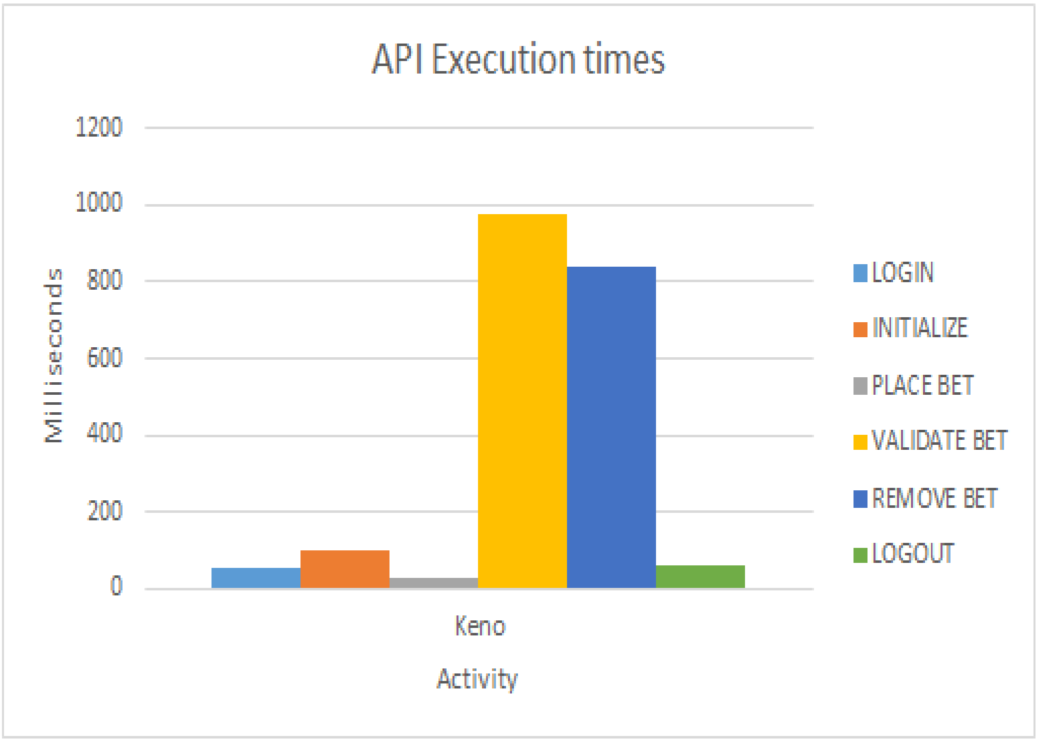


Figure 4.10 Keno API execution times

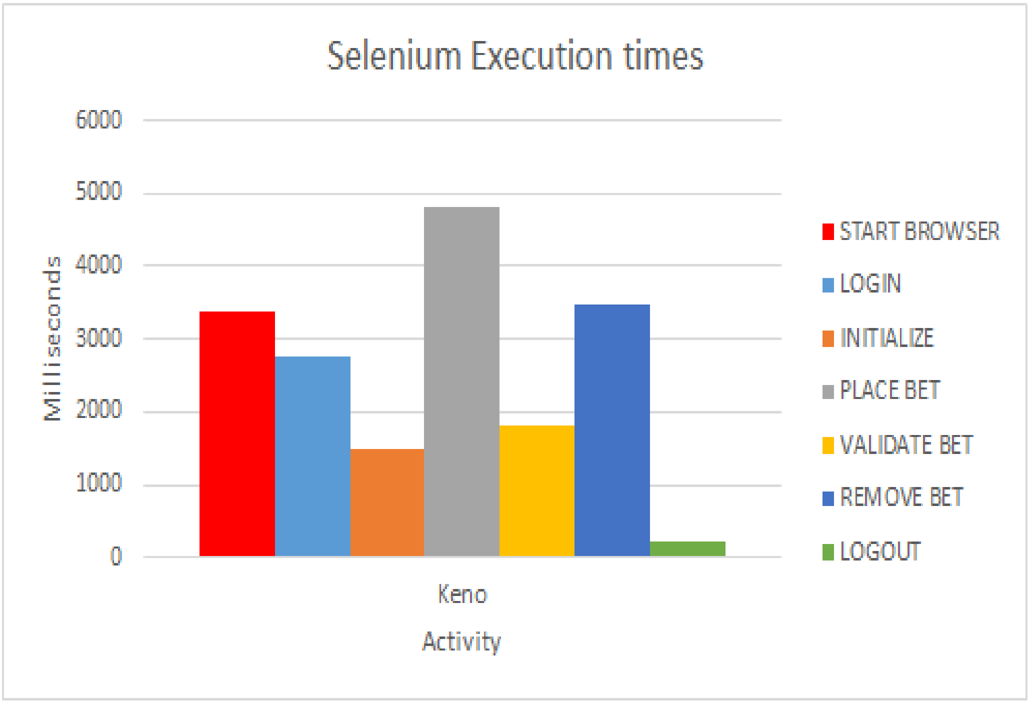


Figure 4.11 Keno Selenium execution times

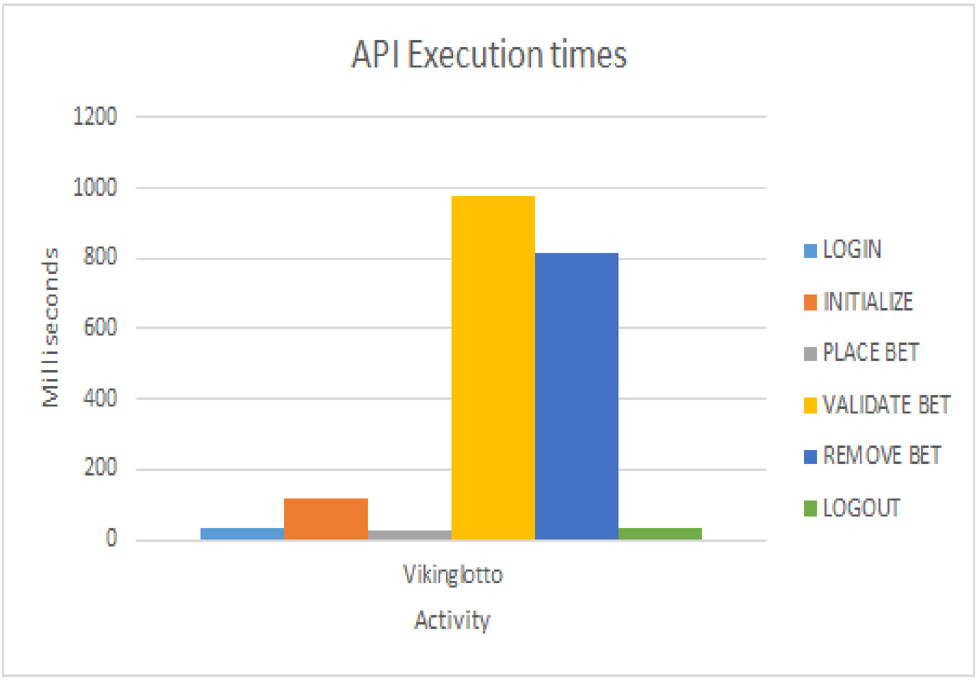


Figure 4.12 Vikinglotto API execution times

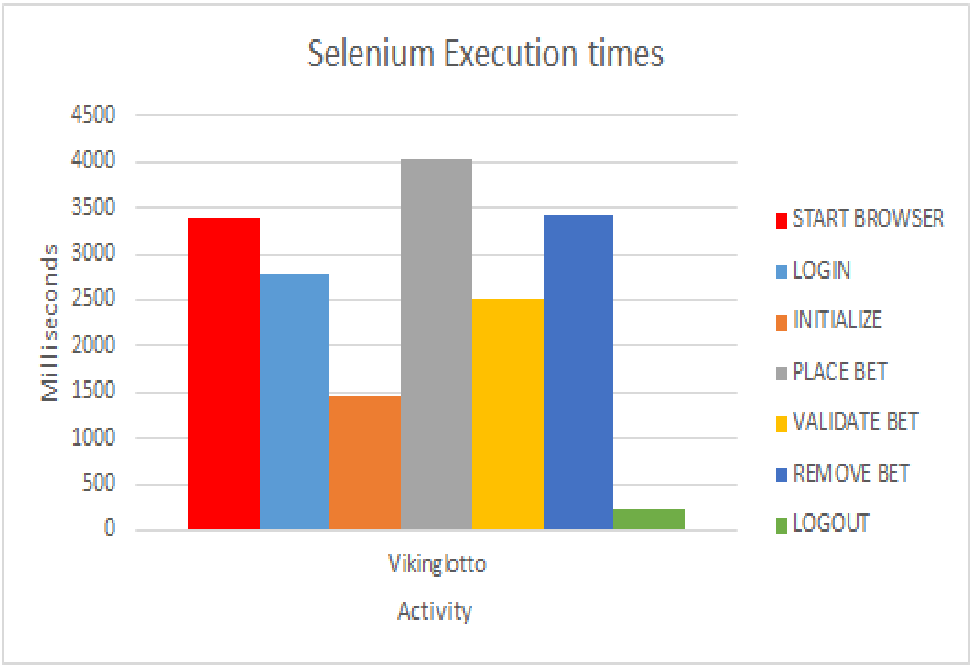


Figure 4.13 Vikinglotto Selenium execution times

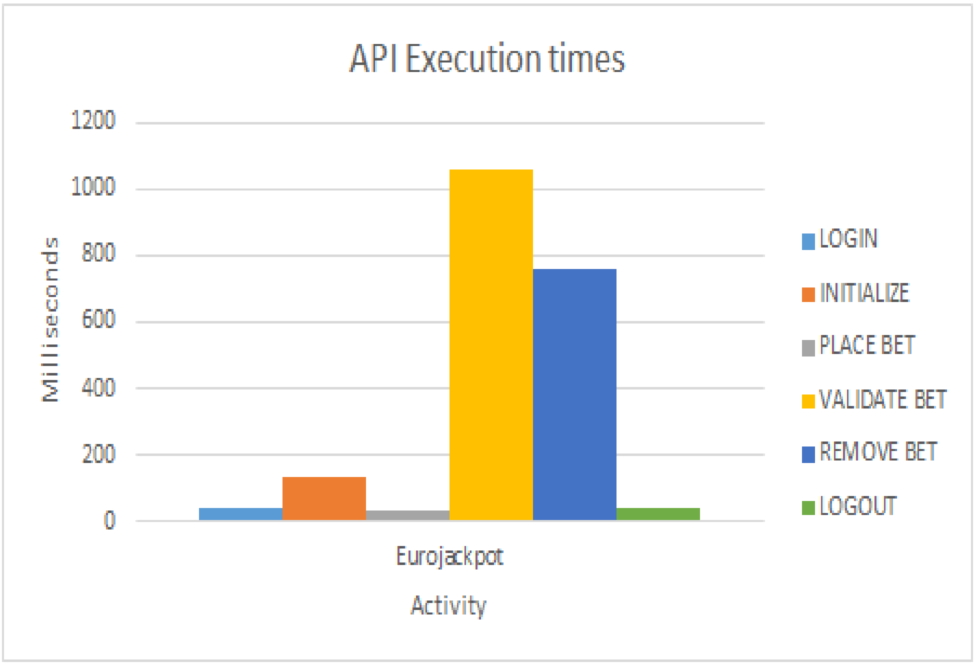


Figure 4.14 Eurojackpot API execution times

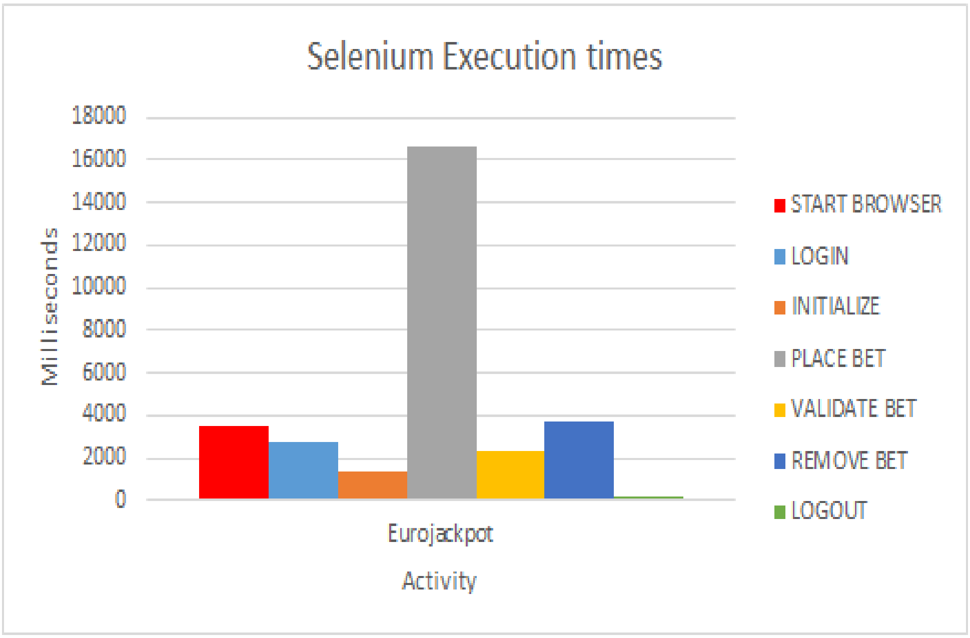


Figure 4.15 Eurojackpot Selenium execution times

Scroll into view (ms)	Set value (ms)	Set kung keno (ms)
122.82	154.22	163.38

Table 4.16: Setting Keno values Average execution times from 100 test executions

Scroll into view (ms)	Set value (ms)
147.78	161.41

Table 4.17: Setting Vikinglotto values Average execution times from 100 test executions

Wait for random values (ms)	Scroll into view (ms)	Select wheel (ms)	Set wheel (ms)
1782.80	29.57	65.19	95.97

Table 4.18: Setting Eurojackpot values Average execution times from 100 test executions

Game	Step 2 (ms)	Step 3 (ms)	Step 4 (ms)
Keno	50.63	63.11	27.23
Vikinglotto	33.08	56.76	26.36
Eurojackpot	35.45	90.00	27.36

Table 4.19: Detailed API execution times Step 2-4 (per API call) from 100 test executions

Game	Step 5 (ms)	Step 6 (ms)	Step 7 (ms)
Keno	28.37	28.23 + 28.37	56.98
Vikinglotto	25.93	26.81 + 25.93	32.43
Eurojackpot	27.13	28.30 + 27.13	38.25

Table 4.20: Detailed API execution times Step 5-7 from 100 test executions

4.2 Flaky test results

The results of the test cases were grouped into 4 categories. Successful, Hanging browser session, Navigational errors, and Functional errors.

A hanging browser connection was detected by means of a timeout when the browser did not respond to any of the selenium calls for 20 seconds.

Navigational errors were detected if the browser was responsive but the element could not be located or accessed for a repeated number (100) times.

Functional errors were detected when the navigation was successful but for some reason the functionality was not provisioned by the system.

4.2.1 Test results over time for Keno

Date	Success Rate (%)	Hanging browser session (%)	Navigational errors (%)	Functional errors (%)
2016 w47	60	25	15	
2016 w48	64	18	18	
2016 w49	72	28		
2016 w50	85	15		
2016 w51	60	10	30	
2016 w52	74	16	10	
2017 w01	82	18		
2017 w02 *	100			
2017 w03	100			
2017 w04	100			
2017 w05	100			
2017 w06	100			
2017 w07	100			
2017 w08	100			
2017 w09	100			
2017 w10	100			
2017 w11	100			
2017 w12	100			
2017 w13	100			
2017 w14	100			
2017 w15	100			
2017 w16	100			
2017 w17	100			

Table 4.21: Average execution result over time for Keno

*) In week 2, 2017 the browser used was switched from Mozilla Firefox to Google Chrome.

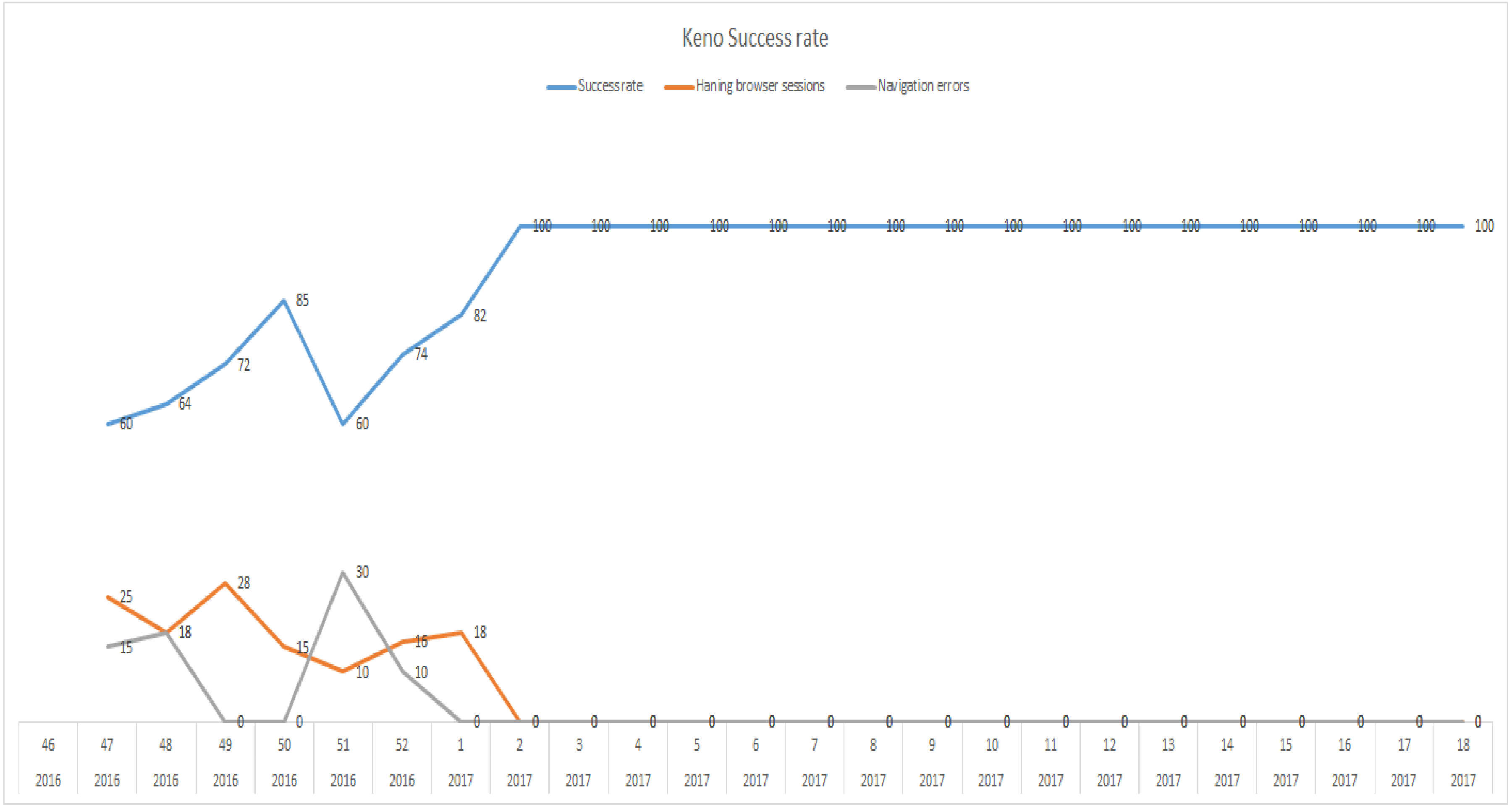


Figure 4.22: Keno success rate over time

4.2.2 Test results over time for Vikinglotto

Date	Success Rate (%)	Hanging browser session (%)	Navigational errors (%)	Functional errors (%)
2016 w46	10	40	30	20
2016 w47	60	20		20
2016 w48	54	26		20
2016 w49	32	28	10	30
2016 w50	18	62		20
2016 w51	47	33		20
2016 w52	70	30		
2017 w01	64	36		
2017 w02 *	80		20	
2017 w03	80		20	
2017 w04	80		20	
2017 w05	80			20
2017 w06	80			20
2017 w07				100
2017 w08	100			
2017 w09	100			
2017 w10	100			
2017 w11				100
2017 w12				100
2017 w13				100
2017 w14	100			
2017 w15				100
2017 w16				100
2017 w17	100			

Table 4.23: Average execution result over time for Vikinglotto
*) In week 2, 2017 the browser used was switched from Mozilla Firefox to Google Chrome.

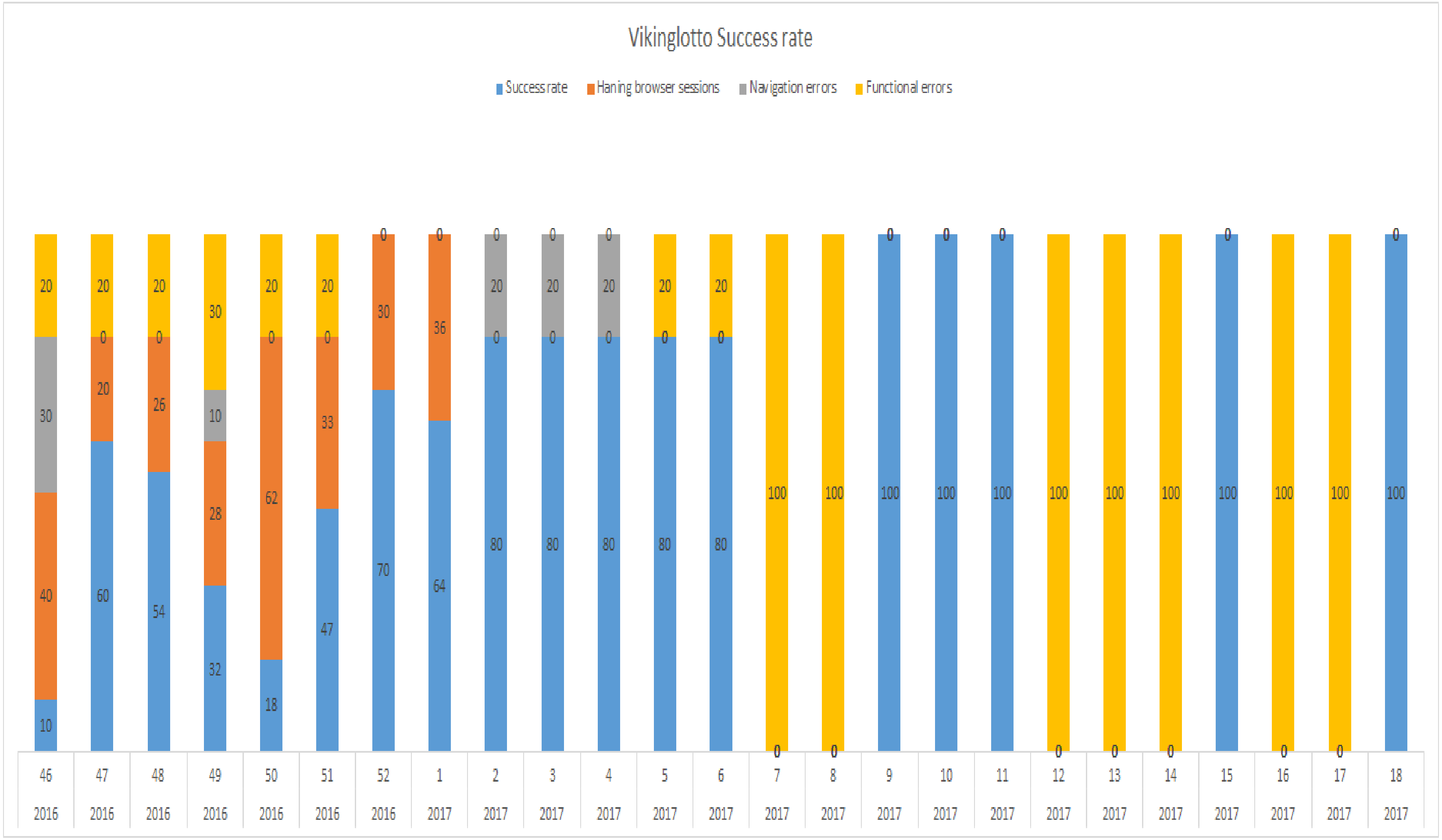


Figure 4.24: Vikinglotto sucess rate over time

4.2.3 Test results over time for Eurojackpot

Date	Success Rate (%)	Hanging browser session (%)	Navigational errors (%)	Functional errors (%)
2017 w04	71			29
2017 w05	71			29
2017 w06	71			29
2017 w07	71			29
2017 w08	71			29
2017 w09	79			21
2017 w10	79			21
2017 w11	79			21
2017 w12	100			
2017 w13	100			
2017 w14	100			
2017 w15	100			
2017 w16	100			
2017 w17	100			

Table 4.25: Average execution result over time for Eurojackpot

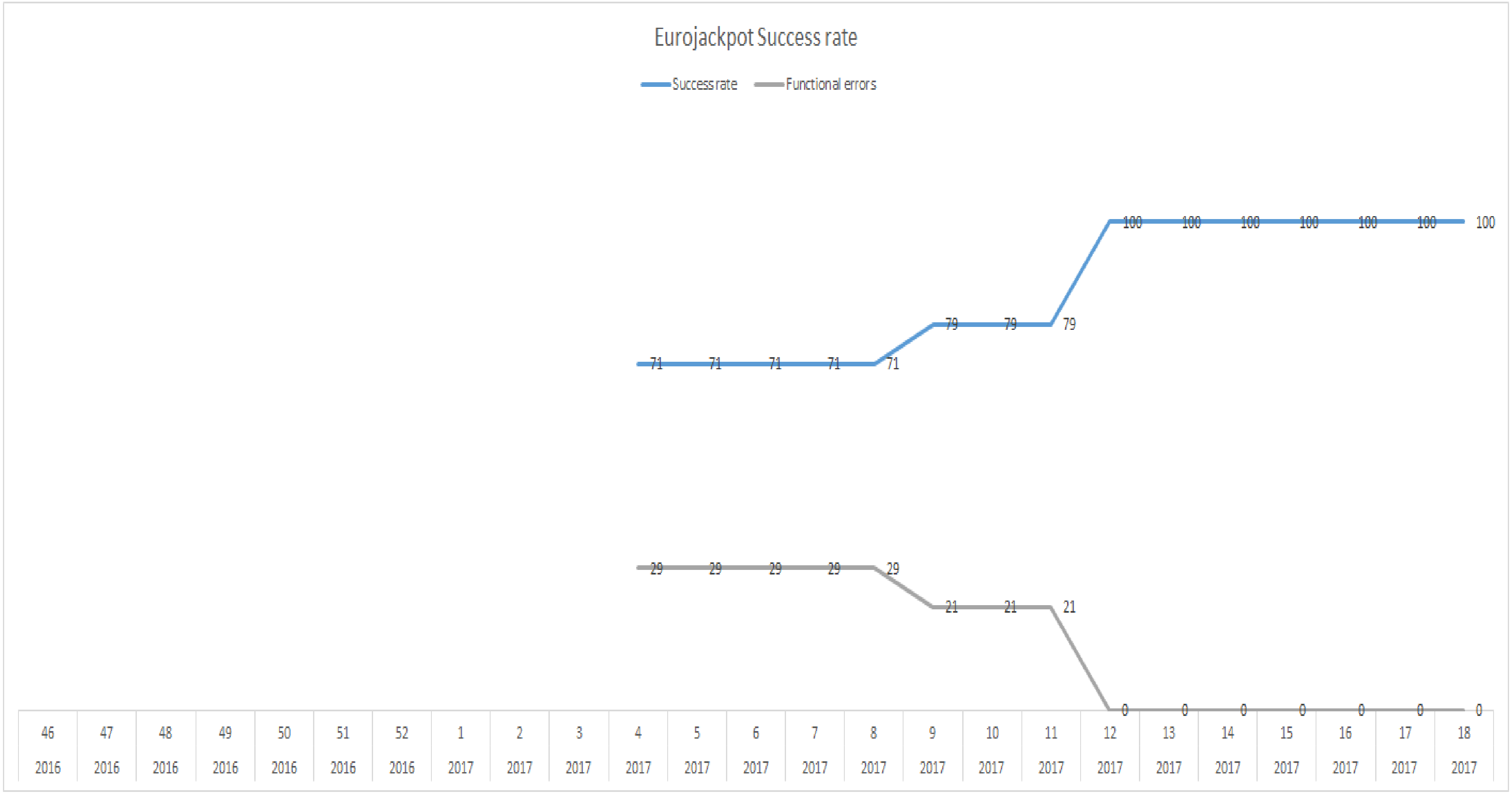


Figure 4.26: Eurojackpot success rate over time

The Functional errors in Eurojackpot were due to a faulty navigation implementation in the FIA tool.

4.3 Modifications to GUI navigation

The classification for a needed modification was that an update to the tool was necessary to locate an element in the GUI, when it previously had worked.

Keno and Eurojackpot were not modified during the time period, however they share common elements with the other games, which were modified during the period. Vikinglotto was the only new games that was designed during the time period when the research was conducted.

Date	Keno	Vikinglotto	Eurojackpot
2016 w46		7	
2016 w47			
2016 w48			
2016 w49		1	
2016 w50			
2016 w51			
2016 w52			
2017 w01			
2017 w02		3	
2017 w03		1	
2017 w04		3	
2017 w05			
2017 w06			
2017 w07			
2017 w08			
2017 w09			
2017 w10			
2017 w11			
2017 w12			
2017 w13			
2017 w14			
2017 w15			
2017 w16			
2017 w17			

Table 4.27: Number of times GUI navigation was updated

5 Analysis

Analysis of the test result was broken down in three different categories, execution times, flakiness and finally issues causing test cases to break.

5.1 Required execution times

The API sequence was very similar regardless of which game that was played and the number of rows being placed in the bet. Because of this the variation in execution times being spent on placing the bet was very small. This can be seen in table 4.4 and figure 4.5 where the test execution times are observed to be close to 2 seconds per test case regardless of which type bet and game that was played. The difference in execution times was between 1977.10 and 2137.88 milliseconds, a difference with 160.78 milliseconds.

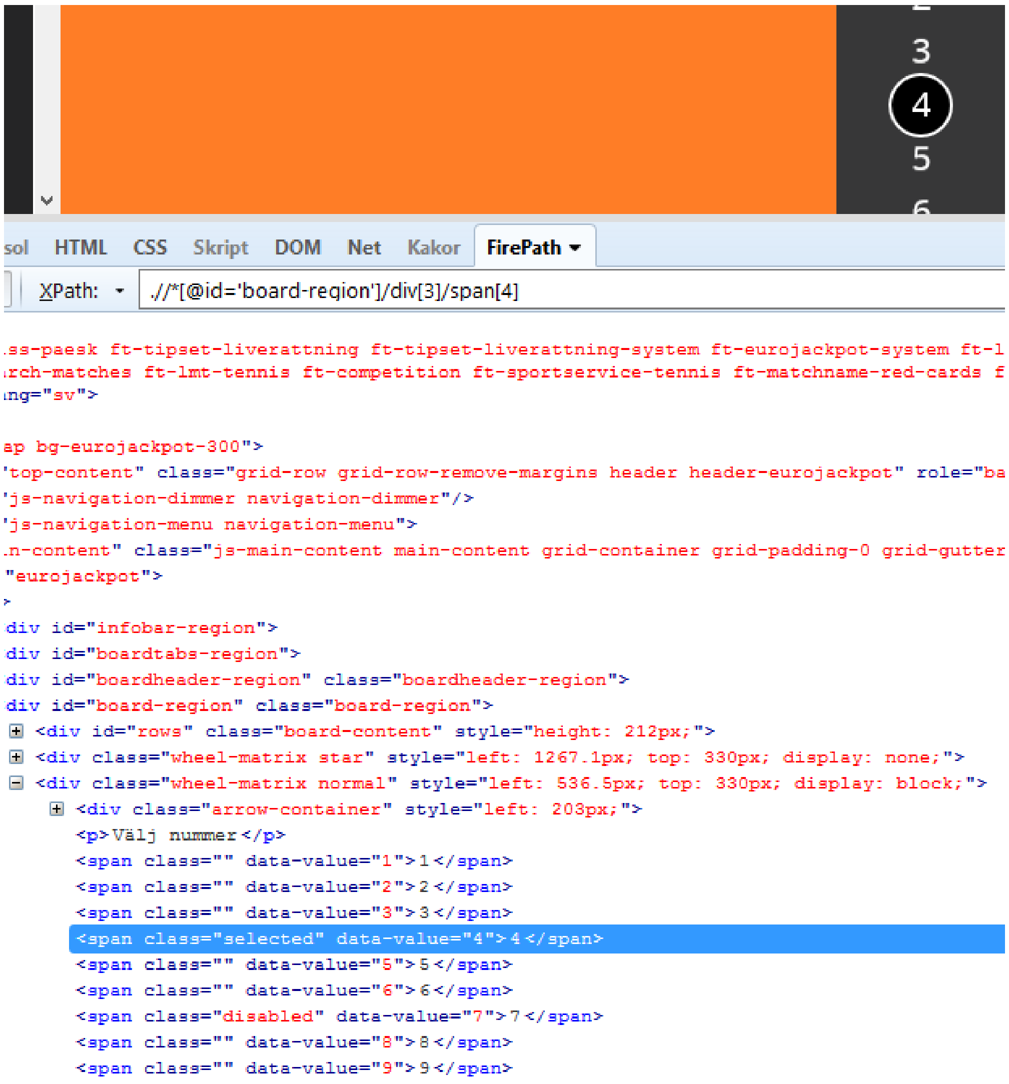
This small difference was derived mainly from two sources. The time taken by the LAN to send and retrieve the requests as well as the internal delay between a placed bet and until it was possible to query the bet. Once the bet had been placed there was a delay in the system until the component responsible for lookup of information on placed bets could access the data. The most of the execution time consumed in the API sequence flow was spent in the validation part, validating that the bet had been placed and that the bet had been removed (figures 4.8 – 4.15).

The GUI navigation on the other hand was very similar in the initial and final parts, with startup, login and navigation to the page where the bet could be placed, as well as the validation and removal of a placed bet (figures 4.8-4.15).

The differences in the navigation were in the part where the bet was placed. Which mainly was affected by the number of elements that needed to be selected. Also the visual effects of the game had to be taken into account. In the games Keno and Vikinglotto there were no significant visual effects when requesting the game to be played. However in the game Eurojackpot there was an effect which randomizes the initial values. This was partly the reason behind the significantly longer time spent placing a game of Eurojackpot compared to Keno and Vikinglotto (figure 4.9).

The time spent on placing a game was also affected by the pattern implementation of locating the web element and scrolling the element into view prior to selecting the element. While this reduced the number of flaky test results it affected the execution time of placing the bet. 29.57-147.78 milliseconds (figure 4.16-4.18) were used in average on moving the object into view where it could be selected. Selecting a single object took between 154.22 – 161.41 milliseconds (figure 4.16-4.18). Each game required a few seconds just to place the bet.

The bigger time difference placing a Eurojackpot game was not only related to waiting for the effect of displaying the random wheels. It was also due to how data could be read in a reliable way. The trustable way to read the value of each wheel was to first select the wheel. Once the wheel was selected the value could be read by reading the attribute value of the class of the wheel (figure 5.1).



```
.//*[@id='board-region']/div[3]/span[4]
```

```
ss-paesk ft-tipset-liverattning ft-tipset-liverattning-system ft-eurojackpot-system ft-l
rch-matches ft-lmt-tennis ft-competition ft-sportservice-tennis ft-matchname-red-cards f
ng="sv">

ap bg-eurojackpot-300">
'top-content" class="grid-row grid-row-remove-margins header header-eurojackpot" role="ba
'js-navigation-dimmer navigation-dimmer"/>
'js-navigation-menu navigation-menu">
n-content" class="js-main-content main-content grid-container grid-padding-0 grid-gutter
"eurojackpot">
>
div id="infobar-region">
div id="boardtabs-region">
div id="boardheader-region" class="boardheader-region">
div id="board-region" class="board-region">
 ⊞ <div id="rows" class="board-content" style="height: 212px;">
 ⊞ <div class="wheel-matrix star" style="left: 1267.1px; top: 330px; display: none;">
 ⊟ <div class="wheel-matrix normal" style="left: 536.5px; top: 330px; display: block;">
    ⊞ <div class="arrow-container" style="left: 203px;">
      <p>Välj nummer</p>
      <span class="" data-value="1">1</span>
      <span class="" data-value="2">2</span>
      <span class="" data-value="3">3</span>
      <span class="selected" data-value="4">4</span>
      <span class="" data-value="5">5</span>
      <span class="" data-value="6">6</span>
      <span class="disabled" data-value="7">7</span>
      <span class="" data-value="8">8</span>
      <span class="" data-value="9">9</span>
```

Figure 5.1 Wheel value locator

The effect of using Selenium was reduced by avoiding the internal delays provided by Selenium. The FIA tool implemented an internal loop that located the element as fast as possible, reducing the execution times as much as possible.

The validation of a placed bet also added to the longer execution times. The approximately one second delay in being able to access the placed bet information as seen in the API cases (table 4.7) resulted in a need to at some occasions reload the full mina-spel page. This page was used to validate that the game had been successfully placed in the Selenium case. If the data was not accessible the first time the page was accessed, the whole page needed to be reloaded.

These issues collectively added up in a significantly longer execution time when Selenium was used.

A comparison between the execution times also needed to consider that when the web browser was used to place the bet, significantly more information was also requested from the system. Effects like greeting messages, fetching the players name, settings etc. was not required in the API case, however when the GUI is used it is a part of the implementation and thus required.

This made it difficult to compare the values in table 4.4 and tables 4.16-4.20 by just mapping them one to one.

Also the tradeoff between speed and reliability that was required to reduce flakiness added additional execution time to the GUI navigation. This resulted up to 10-35 times longer execution times when placing a game using Selenium compared to executing the test case using the API calls. Increasing the execution times from roughly 2 seconds to 20-75 seconds.

5.2 Flaky test results

The selection of tools had a major impact on the test result. Mozilla Firefox and the GeckoDriver are still being developed and as can be seen in tables 4.21 and 4.23, the successful test execution results were improved after the browser was changed 2017 week 2. The same code driving the test cases was used. The only modification was the new interface towards Chrome and its driver. Therefore unquestionably the browser replacement was a major contributor to the increased success rate.

When the instability of the browser was removed the real problems causing flakiness in the test results were more visible, and listed as navigational errors in table 4.21, 4.23 and 4.25. Due to the difficulty to automatically analyze the cause, a screen dump was made when the problem occurred and the screen dump and the logged data was manually analyzed. It showed two main areas of concern. Either the element could be found in the DOM, but for some reason it was not accessible. Or secondly the element was not found in the DOM, indicating that the sequence flow of the navigation was not as expected.

Vikinglotto was a new game and a number of times the whole test suite failed. Mainly due to other testing being conducted in the same test environment. In table 4.21 it is shown as a 100% error rate of the functionality. The test case failed as it was not possible to place the game, as this functionality was temporarily disabled.

The Eurojackpot test cases listed as functional failures were due to missing of implementation in the FIA tool.

5.3 Keeping test cases running

The API sequence was not modified during the data collection, and the cases were always running successfully.

The GUI navigation was only updated for Vikinglotto at a few occasions. In total 15 occasions broke the test case due to GUI elements being replaced (table 4.27).

Eurojackpot and Keno were both existing games and were not redesigned during the time interval when the data was collected, only Vikinglotto was a new product. Even though a number of GUI elements are common for all the existing games the navigation only broke on a few occasions.

6 Discussion

The differences in execution times are not easy to compare between different web applications as each comes with its own set of dependencies. Because of this a comparison with other research on web applications as such would mainly be speculative.

But this research shows that improving test execution times needs to be considered in a wider scope. Some improvements are possible to do by optimization and minor improvements to the navigation in some parts of the test code. However it will not reduce the effects from the time spent on visualization of objects in the web browser. As an example there is approximately a 3 seconds spent on just retrieving the digits randomly selected for a Eurojackpot row. When it comes to testing of the business requirement of placing a Eurojackpot bet this extra delay does not contribute to anything. If the values would be readable in advance to the spinning wheels effect the navigation could disregard to the spinning of the wheels and continue with the execution considerably faster. This shows that the design already needs to take into account how the code is to be tested if significant time savings should be achieved.

The implemented navigation in the FIA tool does not use delays. It rather tries to find the following object as fast as possible. Which is achieved when it fulfills the two requirements of object being located and scrolled into view, then the object can be acted upon (Appendix A.1).

As mentioned the issues need to be evaluated from a wider perspective. However the main issues identified as contributing most to the execution times are:

1. Waiting time for objects to complete visual effects prior to being in a reliable state where interaction with the element is possible.
2. The need to ensure that objects to be interacted on always are in view, where the browser implementation allows for interaction with the element.

These are the top two issues found during the research and also answered RQ1.

In the beginning Mozilla Firefox was the only web browser used, however due to the flaky test results and continuously hangings of the GUI navigation a second browser Google Chrome was also taken into use. The difference is seen as the increase in the success rate as well as the fact that hanging browser sessions are no longer detected. The browser was swapped starting from 2017 week 2 (tables 4.21 and 4.23).

The remaining navigational errors that sometimes cause a test case to fail were analyzed by looking at the screenshots which are made when a fault occurs.

Firstly the reasons behind most of the failures were related to an element being in a state where it could not be selected, as it was covered by another element (figures 6.1, 6.2, and 6.3).

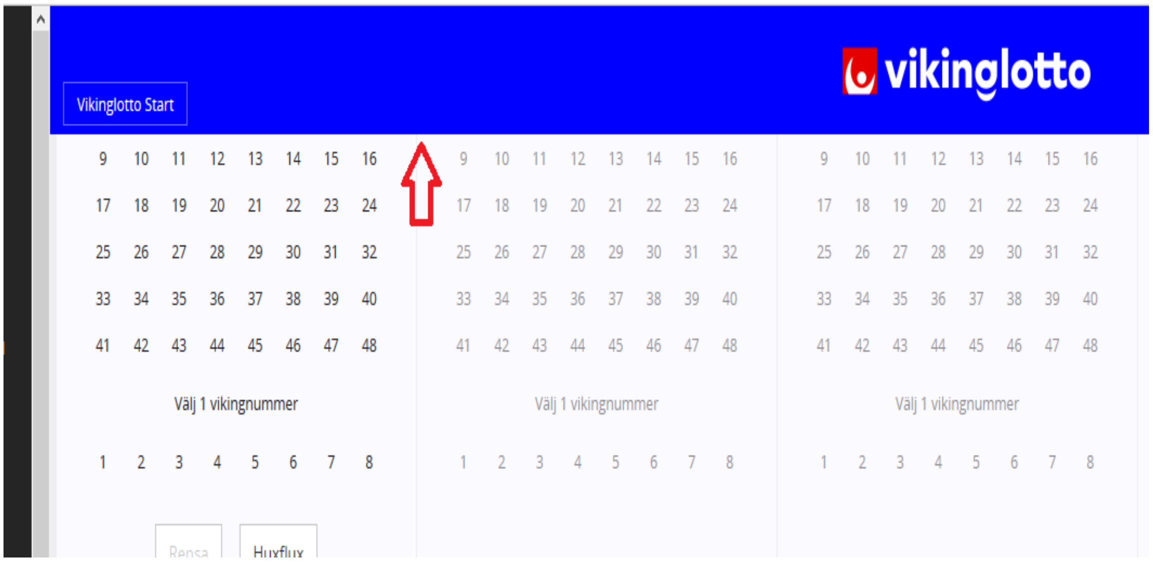


Figure 6.1 Digits 1-8 hidden under the banner

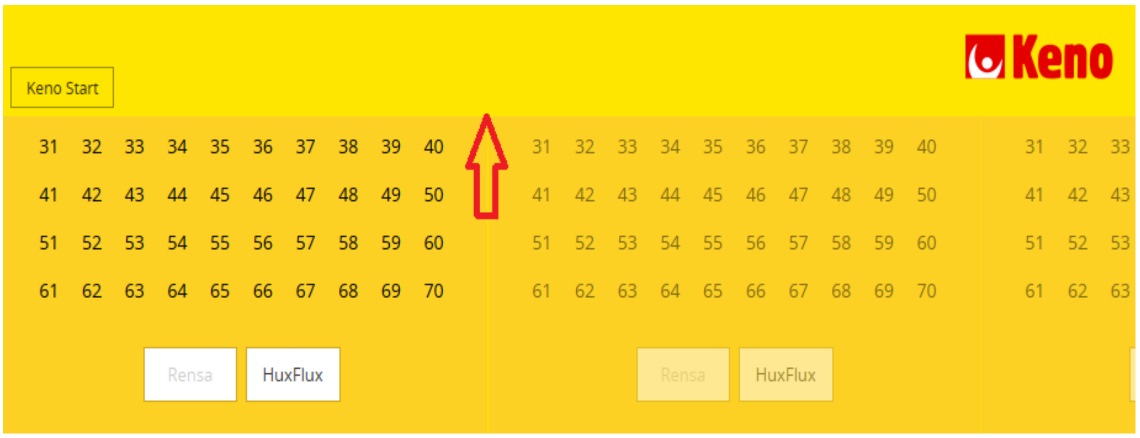
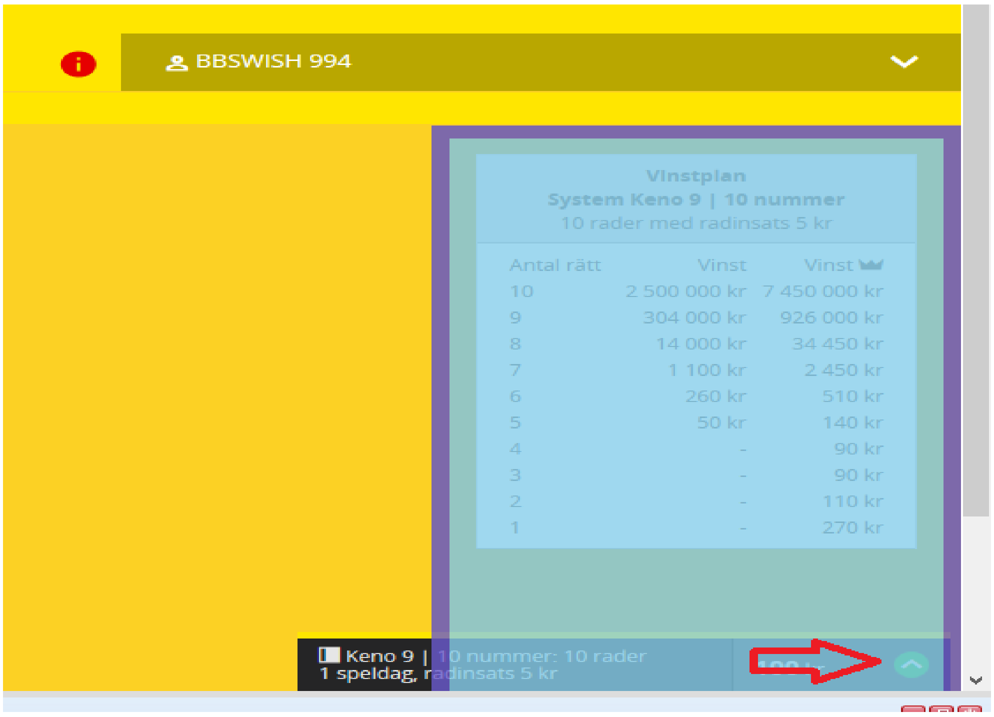


Figure 6.2 Digits 1-30 hidden under the banner



Vinstplan
System Keno 9 | 10 nummer
10 rader med radinsats 5 kr

Antal rätt	Vinst	Vinst
10	2 500 000 kr	7 450 000 kr
9	304 000 kr	926 000 kr
8	14 000 kr	34 450 kr
7	1 100 kr	2 450 kr
6	260 kr	510 kr
5	50 kr	140 kr
4	-	90 kr
3	-	90 kr
2	-	110 kr
1	-	270 kr

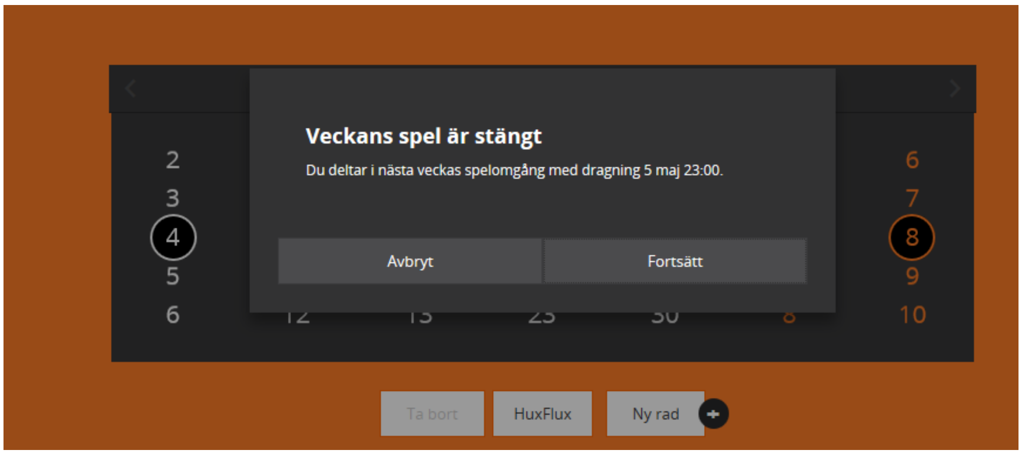
Keno 9 | 10 nummer: 10 rader
1 speldag, radinsats 5 kr

Figure 6.3 Button hidden under the info window

Even though the element is hidden it is still visible in the DOM, which selenium is interacting with.

The solution applied is to ensure that the element is always fully visible and scrolled into view before being clicked. The solution is shown in appendix A.2.

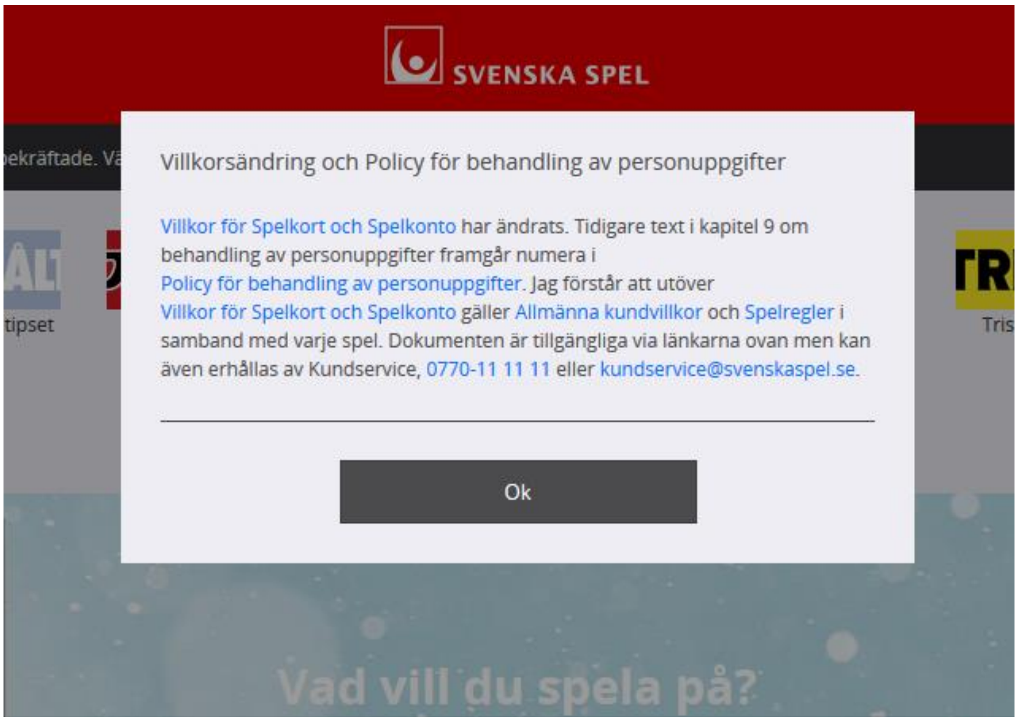
Secondly there are unexpected popups that are not part of the navigational flow which is considered to be the normal flow of placing a bet. A few of these unexpected popups are shown in figures 6.4-6.8.



Veckans spel är stängt

Du deltar i nästa veckas spelomgång med dragning 5 maj 23:00.

Figure 6.4 Popup closed for new bets

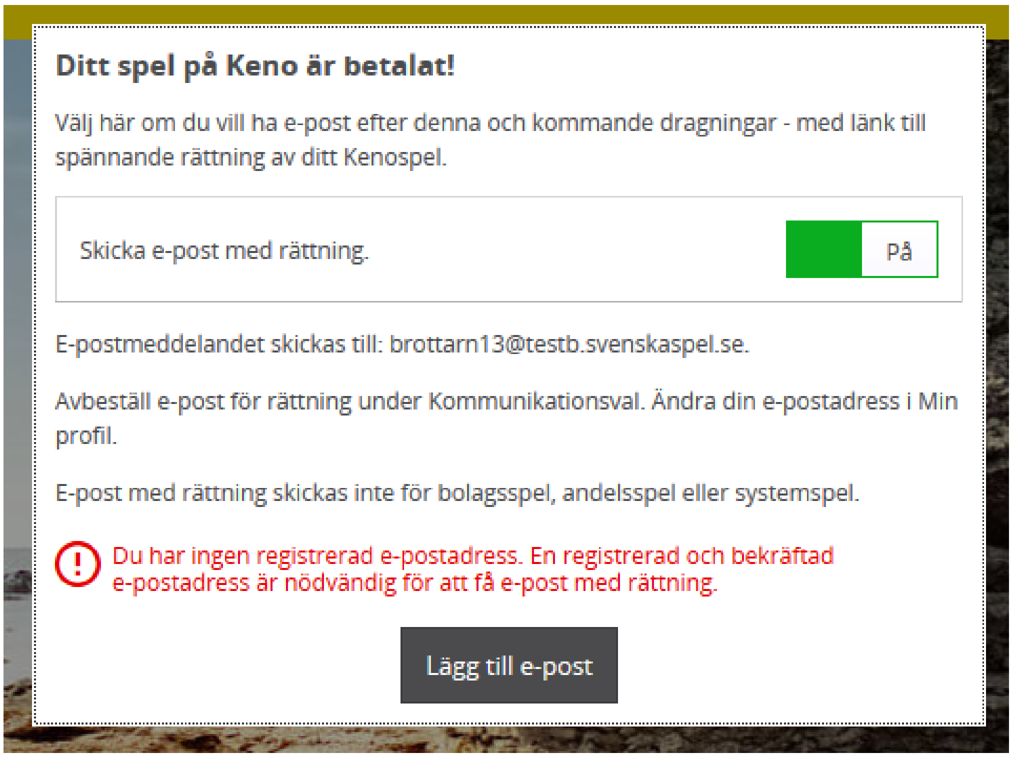


Villkorsändring och Policy för behandling av personuppgifter

Villkor för Spelkort och Spelkonto har ändrats. Tidigare text i kapitel 9 om behandling av personuppgifter framgår numera i Policy för behandling av personuppgifter. Jag förstår att utöver Villkor för Spelkort och Spelkonto gäller Allmänna kundvillkor och Spelregler i samband med varje spel. Dokumenten är tillgängliga via länkarna ovan men kan även erhållas av Kundservice, 0770-11 11 11 eller kundservice@svenskaspel.se.

Ok

Figure 6.5 PUL requirements changed



Ditt spel på Keno är betalat!

Välj här om du vill ha e-post efter denna och kommande dragningar - med länk till spännande rättning av ditt Kenospel.

Skicka e-post med rättning. På

E-postmeddelandet skickas till: brottarn13@testb.svenskaspel.se.

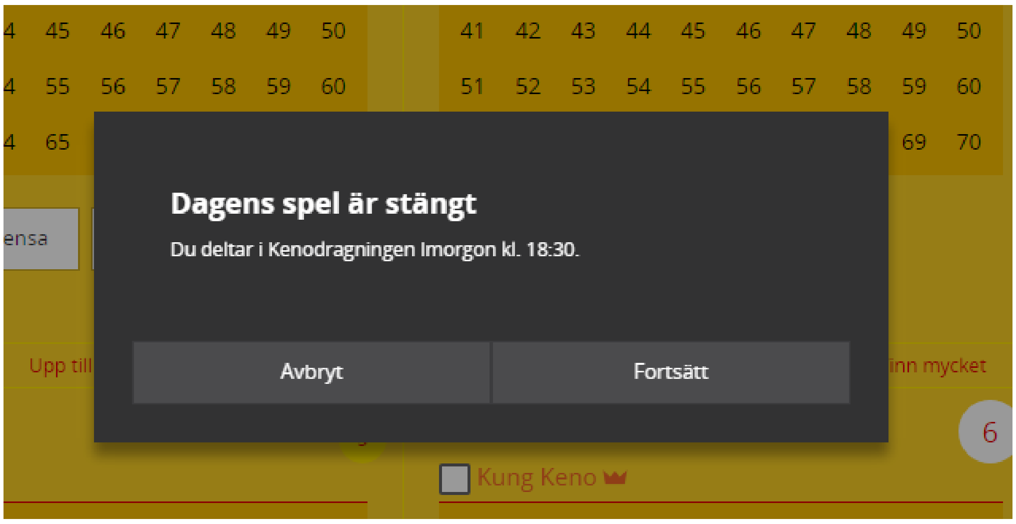
Avbeställ e-post för rättning under Kommunikationsval. Ändra din e-postadress i Min profil.

E-post med rättning skickas inte för bolagsspel, andelsspel eller systemspel.

Du har ingen registrerad e-postadress. En registrerad och bekräftad e-postadress är nödvändig för att få e-post med rättning.

Lägg till e-post

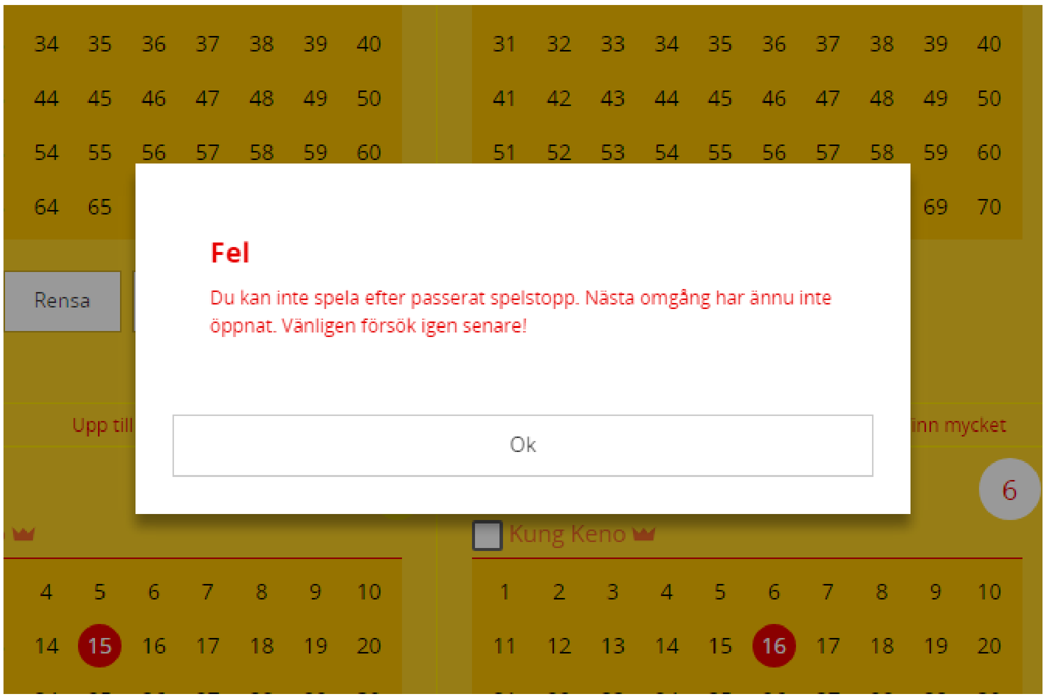
Figure 6.6 Result mail popup



Dagens spel är stängt

Du deltar i Kenodragningen Imorgon kl. 18:30.

Figure 6.7 Keno game closed



Fel

Du kan inte spela efter passerat spelstopp. Nästa omgång har ännu inte öppnat. Vänligen försök igen senare!

Figure 6.8 Keno game closed while placing bet

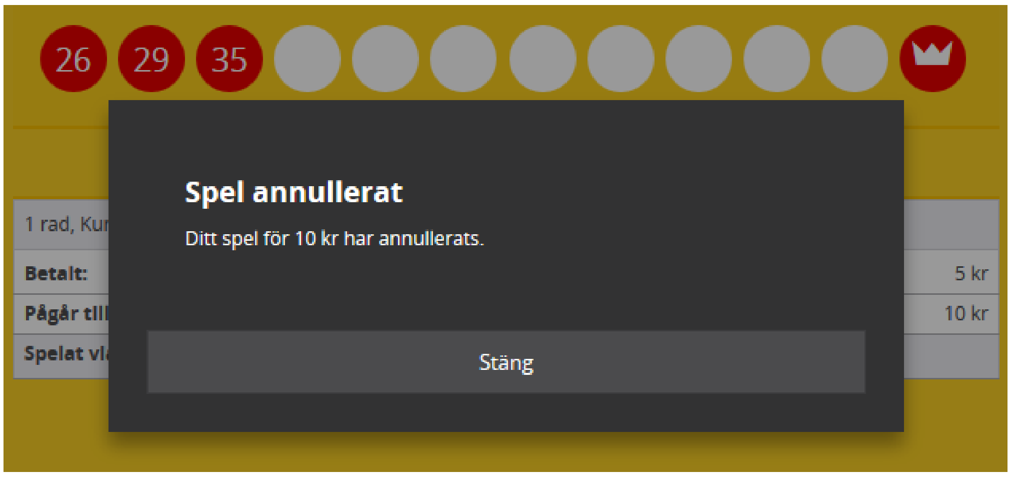
Two measures are applied to limit the effects of unconsidered popups. Prior to starting the test run all users are validated and all flags related to popups are cleared. The timing of when a test run is executed is also implemented to avoid games being closed popups.

The main issues causing flaky test results are:

1. Elements are hidden underneath other objects and cannot be interacted with.
2. Unexpected popups.

After applying corrective actions to these two issues test cases are executed successfully with a 100% success rate, this answers RQ2. This is also partly in line with the research done by Hammoudi et al [12] where one of the issues that broke test cases where Popup boxes causing broken test cases in 4.98% of the cases. The main reason causing failures (73.62%) is due to issues with locating the element, out of these cases attribute not found contributes with 44.51% according to Hammoudi et al [12]. This can be similar to elements hidden, but the research paper does not specifically state that it is due to objects being hidden.

Vikinglotto was the only game being heavily modified during the period of this research. This can be considered as a reason for the required number of changes to the navigation (table 4.4.1) being low. However the implementation of the FIA tool already considered the probability of changes by locating objects on a page using its displayed name rather than a fixed path (figure 6.9 and appendix A.3), as the probability of name changes is less than components being relocated on the web page.



Spel annullerat

Ditt spel för 10 kr har annullerats.

Figure 6.9 Close button

If this contributes to the low number of changes or not, cannot be determined.

Even though RQ3 is addressed, the design decision to use the name of objects to locate their elements shows a good result, but it lacks a reference implementation to compare this design with. RQ3 is addressed, but the result cannot be fully verified. Even though the results are positive, a proper answer to RQ3 cannot be verified with the made research.

7 Conclusion

From this research the conclusion is that test execution times increases 10-37 times and that 100% reliability in test results is achievable when using a GUI driven test approach in regression testing of a Web application when Selenium WebDriver is used.

However the main finding of this research is that regression testing needs to be studied in a wider scope. Decisions made already in the design phase affects the later stages of the project where regression testing is one of the main activities during the product's complete lifecycle.

During software development the characteristics of the software are monitored with its set of parameters with CPU load, Memory consumption, Response times etc. This research shows the need to also monitor the test execution time (RQ1) as a part of the software's characteristics. Not from a user or usability affecting perspective but from a software lifecycle cost perspective, and design decision also needs to consider how testing as a whole is conducted to reduce costs of regression testing.

It is difficult to compare web applications due to the diversity of the functionalities they implement. However when it comes to testing they share the same common problems of ensuring that web elements can be interacted with in a reliable way. This finding is applicable to all web applications. The need for this also increases for web applications that are optimized for a smaller mobile device screen size. In these cases the full page is not necessarily fully loaded. Instead only the top part of the page is loaded and presented to the user as soon as possible. For these kind of devices scrolling elements into view prior to interacting with them increases significantly (RQ2).

The scope and time constraints did not allow for a wider set of products to be tested on. Also most of the components are released and cannot be modified for testing purposes. Testing at the same time with other teams resulted in the product selected to be unusable for longer time periods. If a wider range of selected products would have been developed or were modified RQ3 could have been studied in a better way. Also the choice of using only one browser in the beginning showed to be a bad decision. A plan B should have been implemented sooner.

7.1 Future Research

The method used to locate a web element can significantly affect the execution times. Preliminary research shows that using individual ID locators to find objects are considerably faster. Preliminary trials shows that by replacing the XPATH locator with a unique ID the DOM locating times are somewhat decreased. This confirms the results which Leotta et al. have seen in their case study [16].

However a wider study comparing the different options on how the element is found and scrolled into view is not done. Research in this area could further improve the execution times when the GUI is used.

The second research area which is already mentioned is to look into the impact of taking regression testing and testing needs in general into account already when code is being developed. And how this would impact the cost of regression testing over the entire lifecycle of the product.

References

[1] MSDN Magazine (2013 November issue) ASP.NET – Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. [Online]. Available: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[2] Wikipedia (2017) Single-Page application. [Online].Available: https://en.wikipedia.org/wiki/Single-page_application

[3] ISTQB Glossary regression testing. [Online]. Available: http://glossary.istqb.org/search/regression%20testing

[4] Chittimalli, P.K. and Harrold, M. (2009). Recomputing coverage information to assist regression testing. IEEE Transactions on Software Engineering, 35(4) p.452-469

[5] IEEE STD 610.12 (1990) IEEE Standard Glossary of Software Engineering Terminology. [Online]. Available: http://www.mit.jyu.fi/ope/kurssit/TIES462/Materiaalit/IEEE_SoftwareEngGlossary.pdf

[6] Wikipedia (2017) Document Object Model. [Online] Available: https://en.wikipedia.org/wiki/Document_Object_Model

[7] World Wide Web Consortium (W3C) (2017) What is the Document Object Model. [Online]. Available: https://www.w3.org/TR/WD-DOM/introduction.html

[8] Selenium HQ (2017) [Online]. Available: http://www.seleniumhq.org/

[9] A. Zarrad, "A Systematic Review on Regression Testing for Web-Based Applications" *Journal of Software*, vol. 10 no. 8, p. 971–990, 2015.

[10] Apache JMeter (2017) [Online]. Available: http://jmeter.apache.org/

[11] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "Automated generation of visual web tests from DOM-based web tests" in *Proceedings of the 30th Annual ACM Symposium on Applied Computing 2015 p.775-782*

[12] M. Hammoudi, G. Rothermel and P. Tonella “Why do Record/Replay Tests of Web Applications Break?” in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST) the International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2016 p.180-190*

[13] M. Leotta, D. Clerissi, F. Ricca and P. Tonella “Capture-Replay vs. Programmable Web Testing: An Empirical Assessment during Test Case Evolution” in *20th Working conference on Reverse Engineering (WCRE), 2013 p.272-281*

[14] K. V. Aiya and H.Verma “Keyword driven automated testing framework for web application” at 2014 9th International Conference on Industrial and Information Systems (ICIIS)

[15] M. Leotta, D. Clerissi, F. Ricca and C. Spadaro “Improving test Suites Maintainability with the Page Object Pattern: An Industrial Case Sturdy” in *6th International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2013 p.108-113*

[16] M. Leotta, D. Clerissi, F. Ricca, C. Spadaro “Repairing Selenium Test cases: An Industrial Case Study about Web Page Element Localization” in *6th International Conference on Software Testing, Verification and Validation 2013 p. 487-488*

A Appendix

Code examples used in the validation to overcome observed problems.

A.1 Code example to close Welcome popup

```
//--------------------------------------------------
// CREATE A NAVIGATOR FOR THE MAIN PAGE
//--------------------------------------------------


NavigationMainPage navMain = new NavigationMainPage (driver);

// Close Welcome message
startTime = System.currentTimeMillis();
result = navMain.closeWelcome();
stopTime = System.currentTimeMillis();
stats.increaseMessageCounters("", "WWW_CLOSE_WELCOME_POPUP", result, (stopTime-startTime));
if (result == false){
    System.out.print("\nERROR\tFailed to close Welcome text");
    browser.closeBrowser();
    return false;
}
```

Code A.1 Close Welcome popup

```
public boolean closeWelcome(){

    for (int loopme = 0;MAX_RETRY>loopme;loopme++){

        By closeButton = By.xpath(".//*[@type='button']");
        List <WebElement> closeObject = findElementsList (closeButton);

        for (int loopObj = 0;closeObject.size()>loopObj;loopObj++){

            WebElement closePopUp = closeObject.get(loopObj);

            if (closePopUp.getText().compareToIgnoreCase("Stäng")==0){
                return clickElement (closePopUp);
            }

        }

    }

    System.out.print("\nCan't close the popup as I don't find the button for it!");
    return false;

}
```

Code A.2 Close Welcome popup locator

A.2 Code example to assure that web element is always in view

```java
public boolean playRow (int ruta, ArrayList <Integer> numbers){

    // Align the Row to play
    if (centerRow(ruta) == false){
        return false;
    }

    if (numbers != null){

        for (int loopMe=0;numbers.size()>loopMe;loopMe++){
            boolean result = setKenoNumber (ruta,numbers.get(loopMe));
            if (result == false) return false;
        }

    }

    return true;

}
```

Code A.3 playRow method

```java
// .//*[@id='boardsRegion']/div/div[1]/div
public boolean centerRow (int ruta){

    By kupongRuta = By.xpath(".//*[@id='boardsRegion']/div/div["+ruta+"]/div");

    if (locateElement(kupongRuta)==false) return false;
    WebElement currentElement = findElement (kupongRuta);
    if (currentElement == null) return false;
    clickElement(currentElement);

    return true;

}
```

Code A.4 centerRow method

```java
protected boolean locateElement (By currentXpath){

    for (int retry=0;MAX_RETRY>retry;retry++){

        try {

            WebElement currentElement = findElement (currentXpath);

            if (currentElement != null){

                JavascriptExecutor je = (JavascriptExecutor) driver;
                je.executeScript("arguments[0].scrollIntoView(true);",currentElement);

                return true;

            }

        }

        catch (StaleElementReferenceException e){

        }
        catch (TimeoutException e){

        }

    }

    return false;

}
```

Code A.5 locateElement method

A.3 Code example to close popup boxes

```java
public boolean selectAnnulleraClose(){

    return closePopUp ("Stäng");

}
```

Code A.6 selectAnnulleraClose method

```
private boolean closePopUp (String buttonText){

    final int MAX_RETRY = 600;

    for (int loopme = 0;MAX_RETRY>loopme;loopme++){

        By closeButton = By.xpath(".//*[@type='button']");
        List <WebElement> closeObject = findElementsList (closeButton);

        try {

            for (int loopme2=0;closeObject.size()>loopme2;loopme2++){

                WebElement closePopUp = closeObject.get(loopme2);

                if (closePopUp.getText().compareToIgnoreCase(buttonText)==0){
                    if (locateElement (closePopUp) == false) return false;
                    return clickElement(closePopUp);
                }
            }

        }
        catch (StaleElementReferenceException e){

        }


    }

    return false;

}
```

Code A.7 closePopup method