UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Andrei Proskurin

Adapting a Stress Testing Framework to a Multi-module Security-oriented Spring Application

Master's Thesis (30 ECTS)

Supervisor(s): Mart Oruaas
Raimundas Matulevicius

Tartu 2017

Adapting a Stress Testing Framework to a Multi-module Security-oriented Spring Application

Abstract:

A multi-component system is being build. Three main components are: backend server (Spring application), mobile applications (iOS, Android), customer service web portals. Our main concern is the backend server, because it is the destination of the majority of requests from customer service web portals and mobile applications. It is a multi-module project where all modules communicate to each other. The system is going to be used potentially by hundreds thousands of users with tens thousands of simultaneous usages. Therefore, extensive stress-testing must be conducted. Unfortunately, stress-testing frameworks in the original state are not suitable for the given system. Thus a stress-testing framework must be configured and extended to the point it supports the system's specific protocols and can test all the system's components together. There are numerous of stress-testing frameworks available. Some examples are: Locust, Apache JMeter, Gatling Project. These frameworks differ in terms of coding language, features and core logic. As it is a commercial project, the chosen stress-testing framework must also comply with client's functional and non-functional requirements. Due to stress-testing being conducted only on the backend server component, the selected stress-testing framework must be configured/extended to simulate other components and the required server protocols. The thesis provides a brief comparison of the available stress-testing frameworks based on their features and written code language and define the one which is going to be adapted to conduct the stress-testing within the project and how the adaptation is done. The thesis also points out some of stress-testing frameworks' limitations with techniques to overcome them. Finally, the system is tested using the selected testing framework and the results are presented and validated.

Keywords:

Spring, multi-component, mobile application, concurrency, stress-testing, security, simulation, personal data, framework, extension, Gatling project, Locust, Apache JMeter, functional and non-functional requirements.

CERCS:

P175 Informatics, systems theory

Koormusetestimise raamistiku kohandamine turvalisusele orienteeritud mitmemoodulilise Spring rakenduse jaoks

Lühikokkuvõte:

Programmeeritakse mitmekomponendilist süsteemi. Kolm põhikomponenti on järgmised: põhiserver (Spring rakendus), mobiilirakendused (iOS, Android), klienditeeninduse veebiportaalid. Kõige tähtsam süsteemi töös on põhiserver, kuna see on enamuse veebiportaalide ning mobiilirakenduste päringute sihtpunkt. See on mitmemooduliline projekt, kus kõik moodulid suhtlevad omavahel. Potentsiaalselt hakkab süsteemi kasutama sadu tuhandeid inimesi – kümneid tuhandeid paralleelseid sessioone. Seetõttu tuleb läbi viia süsteemi ulatuslik koormustestimine. Kahjuks on nii, et koormustestimise raamistikud oma originaalseisus ei sobi antud süsteemi testimiseks. Seega, koormustestimise raamistiku tuleb seadistada ning laiendada selleks, et see toetaks antud süsteemi spetsiifilisi protokolle ja võimaldaks testida kõiki komponente üheskoos. Hetkel on saadaval palju koormustestimise raamistikke. Mõned nendest on: Locust, Apache JMeter, Gatling Project. Need raamistikud erinevad üksteisest programmeerimiskeele, eriomaduste ning põhiloogika järgi. Kuna tegu on kommertsprojektiga, peab valitud koormustestimise raamistik vastama kliendi funktsionaalsete ja mittefunktsionaalsete nõuetele. Kuna koormustestimist viiakse läbi ainult põhiserveril, peab seadistama ja laiendama valitud raamistikku, et simuleerida teisi süsteemi komponente ja server protokolle. See töö annab kiire ülevaadet varem mainitud koormustestimise raamistikest eriomaduste järgi, valib raamistiku, mida kohandatakse antud projekti raames koormustestimise läbi viimiseks ning kirjeldab kohandamise protsessi. Samuti toob see töö välja mõned koormustestimise raamistike piirangud ning kirjeldab meetodeid nende ületamiseks. Viimaks, süsteemi testitakse valitud raamistiku abil ning esitatakse ja valideeritakse tulemusi.

Võtmesõnad:

Spring, mitmekomponendiline süsteem, rakendus, konkurrentsus, koormustestimine, turvalisus, simulatsioon, isikuandmed, raamistik, laiendus, Gatling Project, Locust, Apache JMeter, funktsionaalsed ja mittefunktsionaalsed nõuded.

CERCS:

P175 Informaatika, süsteemiteooria

Table of Contents

1 Introduction

In this chapter we are going to provide the motivation for choosing this particular topic (6). Then we are going to present the scope of the thesis (6). Then we will outline the research problems (6) and what is going to be our contribution (6). Finally, we will provide the further structure of the thesis (7).

1.1 Motivation

When designing, developing and testing critical systems it is easy to neglect the quality considerations in the beginning of a project. However, towards a project completion, when code base has grown tremendously compared to the beginning of a project, quality itself as well as quality assurance (QA) can become a serious problem. There are numerous tools available to provide QA aspect and it is often hard to choose something particular, because of project specifications. Another important point is that following a certain set of rules / guidelines from the beginning of a project can potentially be a long-term investment into development and testing quality as well as efficiency.

1.2 Scope

The thesis, first of all, concentrates on comparison of available stress testing frameworks. It provides an insight of essential principles of load and stress testing. Furthermore, it shows the decision process of choosing the appropriate framework for a particular project.

In addition to it, the thesis also describes problems, which were encountered during the QA process of the particular project, specifically a testing framework adaptation to project specifications, application and test design, documentation, testing and development environment setup. The thesis also provides the possible solutions to the aforementioned problems and a set of practices to follow to avoid encountering the concrete problems in the first place.

1.3 Research Problem

As there are numerous free and paid solutions available for complex Spring applications stress and load testing, it has become a problem to choose a suitable tool. There is little chance, that considering a project complexity, any out-of-the-box solution will have all required features to execute necessary QA activities. Therefore, it is required to select the most suitable solution and adapt it to a particular project's specifications.

A QA aspect is often neglected in the project beginning, as the code base is only starting to grow and there is not much functionality to test. However, when a system being developed becomes more complex, the QA becomes a serious problem for a variety of reasons[1]. We are going to research what are the common problems in complex systems' QA processes and what recommendations can be applied to avoid them.

1.4 Contribution

First and foremost, we will provide an insight into choosing a suitable automated testing framework. We will describe and outline the key features of some of the most popular frameworks in the field. We will define certain criteria, based on which a tool should be chosen. Furthermore, we are going to provide a comparison of the selected frameworks based on the mentioned criteria and point out the most suitable tool for our particular case. Finally, we are going to discuss the adaptation process of a

[1]More information in the chapter 4.

testing framework to a project's needs by adjusting and expanding its functionality based on the project's functional and non-functional requirements.

Another important point is presenting the results which we have received by using the selected and adapted testing framework to provide QA in our particular project. We will discuss what problems we have encountered during the application architecture and test design phases and how the provided solutions were organised into a set of recommendations, which are to be followed in order to avoid these problems in the first place.

1.5 Structure

In the 2nd chapter we will provide an insight into some technologies, which were used during our particular project as well as the testing and monitoring tools, that were considered to be used. This chapter will also describe load and stress testing basics and criteria to consider when choosing a stress testing framework for QA. The 3rd chapter will describe the process of the adaptation of the selected framework according to a project's specification. Chapter 4 will provide a set of recommendations / guidelines to follow during an application and test architecture design phases based on our experience in the particular project. The goal of chapter 5 is to theoretically validate our results based on the opinions of experts in the software development, engineering and QA fields. Chapter 6 is going to provide a summary of the thesis, as well as discuss the results, limitations and potential future work.

2 State of the Art

In this chapter we will first of all get familiar with the application's framework in order to decide and outline the testing methodology (8). We will also introduce the tools from which we will choose the stress testing framework to be potentially used for the application testing (8, 9, 9). We will then determine the approach to the testing process (10) on which basis we are going to select the most suitable testing tool for our particular case (11). We will finally provide some of the testing framework limitations as well as system monitoring techniques and tools (12).

2.1 Application and Testing Technology Background

2.1.1 Spring Framework

The Spring Framework is a Java-based platform which provides infrastructure support for Java application development. It's features are organized into about 20 modules, which are divided into groups2. The release of the Spring Framework first version was in the year 2004. Java 8 features were entirely supported in the Spring framework since the release of version 4.0 in the year 2014.

The Spring Framework allows Java developers to deliver straightforward, portable and adjustable JVM-based applications which can be deployed as standalone, on a cloud and in an application server [1]. This is achieved by sharing processes – Spring provides infrastructural support at the application level and removes dependencies on specific deployment platform which allows development team to focus on application's architecture. Every single deployment platform is supported by the Spring Framework with an extensive programming and configuration architecture which is the key to the state-of-the-art Java-based enterprise grade application development.

The Spring Framework key features are:

- aspect-oriented Programming including Spring's declarative transaction management
- dependency Injection
- Spring MVC web application and RESTful web service framework
- foundational support for JDBC, JPA, JMS

Since the first version release, the Spring Framework has become a common choice for Java-based enterprise application development and a mandatory expertise for Java development teams [2].

2.1.2 Locust

The Locust is an open source load and stress testing tool. It allows to simulate millions of concurrent users' behavior from a single machine as well as distributed over numerous machines. The major Locust feature is the support of Python programming language in all test code which is important to many[3] developers [3].

The Locust main objective is to load a web resource (or other system) in order to figure out the number of simultaneous users that can be handled by it. This is achieved by instrumenting test users (locusts) to follow a pre-programmed scenario on the given resource. Each step is monitored and can be viewed in real-time using the Locust application's web-based user interface.

[2]Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging, and Test. ADD REFERENCE

[3]Python is amongst 5 most popular coding languages (TIOBE, IEEE Spectrum, CodeEval).

The Locust is an event-based application. However, contrary to the majority of event-based applications, it utilizes gevent[4] library to provide light-weight processes instead of callback usage which allows the support of thousands of locusts on a single machine. Each Locust test user has its own process which allows to code comprehensive test scenarios without using callbacks [4].

The Locust Framework key features are:

- writing test scenarios using Python programming language
- support of hundreds of thousands of users
- web-based UI
- hackable

Providing valuable statistics is an important feature, which has to be present in each load testing framework to allow users to conduct application analysis in order to point out performance bottlenecks. The Locust statistics are insufficient or absent and do not provide useful information except for the request response times. Data visualization is non-existent. Furthermore, it is problematic to retrieve an error response details apart from a response status and perform non-HTTP / non-RESTful requests [5].

2.1.3 Apache JMeter

The Apache JMeter is another open-source load and stress testing solution. In contrast to the Locust, the Apache JMeter is a Java-based framework. Nevertheless, the goal remains the same – to load test application's functionality and provide performance analysis. The first version originally supported testing only web applications, but since its release it has been further developed to test other systems[5] [6].

The main purpose of Apache JMeter is to generate extensive load on a server or any other supported object in order to evaluate performance and detect bottlenecks under various circumstances (e.g. different load). It also provides data visualization tools (graphs) to conduct performance analysis of a tested application [7].

Apache JMeter key features are:

- Conduction load, stress and performance testing using the following protocols: HTTP(S), SOAP, REST, FTP, JDBC, etc.
- Can be used in different environments.
- Implemented completely in Java.
- Enables concurrency by using threads and allows to separate thread groups.
- Careful GUI design allows faster Test Plan building and debugging.
- Test results can be preserved as cache and analyzed or replayed later.
- Core has a high degree of configurability and extendability.

2.1.4 Gatling Project

The Gatling is another flexible and efficient load testing framework. It has user-friendly, straightforward design and great multi-user performance compared to the Apache JMeter framework.

---

[4]A *coroutine*-based Python networking library that uses *greenlet* to provide a high-level synchronous API on top of the *libev* event loop.

[5]Webservices (SOAP/REST), Web dynamic languages - PHP, Java, ASP.NET, Files, etc. -, Java Objects, Data Bases and Queries, FTP Servers.

In contrast to the Apache JMeter and the Locust frameworks, the Gatling tests (so-called simulations) are based on the Scala programming language [8].

The Gatling exceptional HTTP protocol support makes this framework a good choice for HTTP server load and stress testing. Moreover, the Gatling core engine is flexible and fully supports other protocols implementation.

The main motivation for Gatling project was having programmable and system resource efficient tests. Due to the usage of the specified DSL, the test design is user-friendly. As test scenarrios are written in code, they are highly maintainable and can be handled by a VCS.

In contrast to some other stress testing frameworks (e.g. Apache JMeter), which present virtual users as threads, Gatling uses a message representaion - each user is a separate message. This approach provides a better scaling, does not stress the system, on which tests are executed, and allows to generate excessive amount of load (thousands of concurrent users). In addition to it, Gatling automatically generates HTML reports in the end of each simulation, unless configured otherwise.

The simulation details, such as a number of failed and passed requests, are presented on-the-fly using the Graphite protocol, which highly configurable, widely supported and used for system monitoring and analytics [9].

The Gatling Project key features are:

- Is an open-source software.
- Written in Scala language.
- Concurrency provided by Akka framework.
- Network layer is based on Netty framework.
- Is fast, responsive and light-weight.
- Ability to generate reports in HTML format.
- Ability to record test scenarios using specialized GUI.
- Domain specific language (DSL), which simplifies writing test code.

2.2 Load Testing Principles, Aspects & Guidelines

The main concept of an application load testing is to simulate the actual work and processes that are going to be executed by the real users with artificial or virtual users. The idea is set a performance goal and generate an initial (usually low) load on a target system / application and increase it step-by-step over time until the goal is achieved or the targeted object reaches its limits. The latter means that a bottleneck was hit. After this point throughput stays on the same level (or decreases), request response times increase and functional errors appear [10].

During a load testing process, while the load level increases, it is important to monitor critical system parameters as well as certain load metrics, such as throughput, response times, load test completion. It is necessary to keep these things in mind in order to understand how the target system behaves under a particular load. Another important point is to monitor the injector machine (a machine, which used to inject load on a target system by adding virtual users / threads) performance during load test execution. This allows to ensure that the injector machine is not a bottleneck factor in our target system performance. Close to 100% CPU or memory utilization levels indicate that an injector machine is overloaded and might cause performance test result unreliability.

It is crucial to ensure a target system / application stability before conduction load / performance testing. The reasons are:

- Despite a target system's functional stability, it might have code / design problems, which can cause bandwidth limitations.

- In case a target system uses relational database (e.g. SQL based), there is a chance that some SQL procedures / requests are poorly designed and might cause delays.
- An example of bad system design is an excessive amount of conversations between different system layers, which cause latency and bandwidth problems.
- There is a chance a target system returns obscure under a certain condition. Although a couple of them may not cause performance issues, considering increasing load of performance testing, hundreds thousands of these definitely may.

Additionally, quality performance testing requires thorough planning and time estimation. Activities to consider when planning time are:

- test environment setup and configuration
- injector machines setup and configuration
- use cases analysis, identification and scripting
- test data preparation
- problem solving

It is critical to take into account that tests usually depend on a system version - they perform a set of requests and expect certain responses, which were defined based on an application version available at the moment of test scripting. An application new version release might change API to a point where tests completely or partially fail and must be re-scripted from scratch in the worst case scenario. Thus, it is obviously important to perform load testing on a consistent code release opposed to an actively developed application.

Moreover, it is necessary to take into account an environment to be used for load testing. The best solution would be to have the exactly same test environment as on the client's production server, as it would guarantee test results' reliability. Performance testing accuracy also depends on particular key performance indicators (KPIs), which are part of non-functional requirements. These are:

- availability / uptime
- concurrency
- throughput
- response time

All performance testing scenarios' workflow derive from use cases, therefore they have to be properly analysed and identified. They have to represent a target system's critical activities, which are to be conducted by an average user on a daily basis. The goal of stress testing is not to check whether application functional requirements have been fulfilled, but to assess it from a "performance under certain load" viewpoint in order to reveal issues, that are caused by concurrency problems, bandwidth limitations, slow code segments or poor configuration [11].

2.3 Choosing the Appropriate Tool

Web applications are currently widely spread. Thus, the vast majority of testing tools has HTTP(S) support. However, client end web design specificity (e.g. extensive use of JavaScript, JSON, Microsoft Silverlight, etc.) might be a stress testing framework limitation and should be taken into consideration. Overall things to take into account when choosing a suitable testing tool are:

- Protocol support - a tool must enable the communication between application layers by supporting protocols, required by a target system.
- Licensing model - a revenue model defined by a tool vendor. These models usually depend on the amount of generated load and additional features, which a tool supports (e.g. protocols, plug-ins, monitoring, etc.)

- Scripting effort - ease of expanding a tool to support specific features as well as a level of complexity and team skill, which is required in order to implement test cases.
- Solution versus performance testing tool - a choice between a tool, which is only capable of generating certain load, and a complete solution, which provides other useful features, such as monitoring, report generation, logging, etc.
- In-house versus outsources - a choice whether a target system's development team is going to use, expand and configure a stress testing tool, or another team is planned to be contracted to execute these processes.
- Alternatives - in case a web application is to be tested, it is worth to consider using a Software as a Service(SaaS) application instead of a dedicated testing tool.

In case of our project we are also taking into consideration the client's non-functional requirements document, which outlines several specifications that a considered load testing tool must follow:

1. Open licensing model – allows unlimited use of the product from the vendor in unlimited amount.
2. All protocol support – only one testing tool (out of application code layer) must be used therefore it must support all of the application client protocols.
3. Scripting effort – the testing codebase must be forwarded to the client and should support straightforward extension and maintenance.
4. Additional features – the testing tool should support the application's build mechanism, automatic test report generation, test case execution real-time monitoring.
5. Programming language – the test tool must be run on Java virtual machine (JVM).

JMeter comes with a GUI, which may seem a good feature in terms of the simplification of use case scripting. However, testing scenarios tend to grow larger and become more complex, thus difficult to maintain and extend using the point-and-click interface JMeter provides. Secondly, JMeter is thread-bound. This means that every injected virtual user is a separate application thread. Needless to say, benchmarking thousands of users on a single machine becomes unfeasible.

Locust solves the mentioned problems and is also capable of running load tests distributed across multiple injector machines. The testing scenarios are scripted Python. Locust also uses a light-weight processes approach for virtual user injection, opposed to JMeter resource-heavy threads, which significantly decreases the load put onto injector machines.

Although Apache JMeter, Locust and Gatling provide similar basic functionality, the included features and main logic of how virtual users are injected are different. The reasons, which motivated to use Gatling project the most are based on the criteria presented above. These are:

- Protocol support - basic HTTP(S) client is present, and the core is highly expandable, which allow additional features implementation.
- Licensing model - Gatling is an open-source stress testing tool with open licensing model, which allows generating unlimited load.
- Scripting effort - Gatling DSL simplifies test scenarios scripting. As well as that, the scenarios are self-explanatory and can be handled by VCS as production code.
- Testing solution - Gatling package has a lot of useful features, such as report generation, live monitoring, jdbc feeder, etc. Gatling also integrates with Jenkins and Gradle with the help of certain plug-ins.

2.4 Monitoring

Automated testing framework are powerful tools, which can be extended to a certain degree. However, their functionality has limitations and does not allow monitoring critical system parameters

during test executions. This is especially important during load and stress testing when searching for performance bottlenecks (2.2). This could be overcome by using certain monitoring[6] techniques and tools.

2.4.1 Analysing JVM Critical Parameters

Excessive memory usage is one of the most common Java application problems. It can be caused by issues within system garbage collection (GC) processes. As a result response times increase as well as an application becomes instable and unresponsive. Therefore, GC must be monitored in order to guarantee application stability and performance [12]. It is important to investigate the following aspects:

- Memory pools (Eden, Survivor, Old) utilization. In most cases, excessive memory usage causes abnormally high GC activity.
- Increasing application memory usage despite GC indicates a memory leak, which will eventually cause an application to run out of available memory. To find the leak, a memory heap analysis is required.
- Long response can be caused by a condition when there is a high number of young collections and a growing old generation. This usually means that the young generation memory pool cannot hold such amount of objects.
- The condition of the old generation utilization levels increasing and decreasing, but staying within the same limits without rising, is usually caused by object being unnecessarily copied from the young generation to the old generation. This is usually caused by insufficient young generation memory pool, high *churn*[7] rate, or transactional memory shortage.
- Excessive GC activity levels usually have negative impact on a system CPU usage. This should not affect response times, unless there are suspensions[8]. These events must be monitored taking application response times into account.

Long response times can sometimes be caused by heavyweight method calls. Therefore, we need to monitor total time spent invoking methods including internal method calls in correlation to the application response time. It can require methods which do a lot of processing and/or database communication to be further optimized.

Database connection pool and JMS connection pool figures, such as number of active and idle connections, also need to be monitored. This can result in identifying connection management issues, such us leaving unnecessary connections open, which will eventually lead to memory leaks and/or connection pool shortages.

Large amount of active threads can also result in memory shortages**.** High amount of active threads might indicate application thread management issues. Too many threads may slow down the application as well as the entire server. In this case, a thread dump analysis is required. It is used for analysing what processes an application threads executed at a certain point and what was the state of

[6] System monitoring – retrieving critical system information (e.g. CPU and memory usage) after a certain time interval (usually configured).

[7] Rate of object allocation

[8] *Stop-the-world* events, which stop application threads

each thread. Thread dump analysis done in intervals helps to diagnose application execution problems such as *thread deadlocks*[9].

2.4.2 Monitoring Tools

2.4.2.1 VisualVM

VisualVM is a monitoring tool, which, using the integrated GUI, provides information on multiple Java applications' critical parameters while they are running on JVM. Using VisualVM it is possible to monitor application overall memory utilization as well as GC figures (different memory pools). We can also use it to check a thread count, view detailed statistics and create thread / heap dumps if needed. VisualGC plug-in is recommended to monitor GC because it provides all important statistics combined in one view. These statistics are:

- Metaspace, Old, Eden, Survivor memory pool space status
- GC number, duration, reason

General useful data, such as CPU usage in correlation to GC activity, total heap usage, loaded classes number and active threads number can be found under the *Monitor* tab.

We can also use profiling / sampling to get information about time and memory consuming code segments using profiler / sampler tools respectively. Profiling is more accurate than sampling, but has a higher performance impact. VisualVM profiler works by "instrumenting" all of the methods of code. This adds extra bytecode to methods for recording when they are called, and call execution times. VisualVM sampler, however, takes a dump of all of the threads on a fairly regular basis, and uses this to work out how roughly how much CPU time each method spends. VisualVM also allows to create heap and thread dumps as well as use an integrated thread dump analyser. There are various other tools available for heap and thread dump analysis, such as Memory Analyzer (MAT), IBM Thread and Monitor dump analyser.

2.4.2.2 Jolokia

Jolokia acts as a bridge between application JMX and HTTP interfaces. It enhances standard JMX monitoring capabilities and allows to request information on important parameters using HTTP. It can be used in collaboration with such tools as:

- Curl (or other) simple http GET/POST requests
- HTTP client polling Jolokia and building graphs
- Hawtio dashboard
- Zabbix monitoring server

2.4.2.3 Oracle Internal Tools and P6Spy Framework

Poor database performance is often the cause of low overall application response times. In order to improve it, SQL queries' statistics have to be collected and analysed. As for collecting the data, there are numerous tools available such as Oracle system V$SQLAREA table or external frameworks (e.g. P6Spy). After the slowest (the longest execution time) SQL queries are found, it is possible to analyse them using Oracle's EXPLAIN PLAN statement.

[9] A condition when a process cannot use some resource, because it is used by another process. This means that the former process cannot proceed with its work and has to wait.

P6Spy framework is designed "hijack" a database connection and log incoming / outgoing data without application code changes. The logging is conducted using the P6Log distribution, which allows to log Java application JDBC transactions.

Oracle EXPLAIN PLAN statement is used to provide execution plans[10] selected by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements [13]. It is important to focus on the following EXPLAIN PLAN metrics:

- Operation – the conducted internal operation name.
- Options - a variation on the operation.
- Object name – table / index name.
- Position – the first output row position value indicates the optimizer's estimated cost of the statement execution; other rows show the position relative to the other children of the same parent.
- CPU cost – estimated CPU cost of the statement execution.
- IO cost – estimated I/O cost of the statement execution.

If an operation has high CPU / IO cost, then it has to be optimized. Object name gives us a hint about what table the operation was done on. Operations and options help to identify what might a problem be (e.g. TABLE ACCESS FULL means that the whole table is being scanned for some particular value and may result in high CPU / IO usage). It might be also required to view an execution plan for a particular explain plan which provides a more generalized view of the explain plan key points as well as the SQL query execution plan. After analysing queries explain and execution plans it is clear whether SQL query code and/or database indexes need to be optimized in order to increase execution performance.

2.5 Summary

In this chapter we introduced the Spring framework, the backbone of our application, as well as the stress testing frameworks considered for the application testing – Locust, Apache JMeter and Gatling. In addition to it, we defined the approach to the load testing process, the guidelines, the considerations to follow and the metrics to track. We then performed a basic comparison of the chosen stress testing frameworks and identified the most suitable tool for our particular case. Finally, we discussed some stress testing frameworks' limitations and how they can be compensated by using certain monitoring techniques and tools.

In the next chapter we are going to extend the chosen stress testing framework to support our application testing. We will then apply the defined testing methodology to our application testing process and outline the results.

[10] A set of actions Oracle performs to run the statement

3 Test Tool Integration

In this chapter we are going to provide the background to the system being tested using the chosen testing framework (3.1). Moreover, we are going describe the testing framework's functions, users and expectations (3.2). We will finally describe the adaptation process by providing a set of required extensions / improvements to the testing framework based on a project's specifications (3.3).

3.1 Background

Smart-ID application is going to be used for user online authentication, which allows to use government and bank services on a mobile device. The application is going to be deployed on a remote server. Mobile applications and self-service portal will send requests and receive responses from the Smart-ID application.

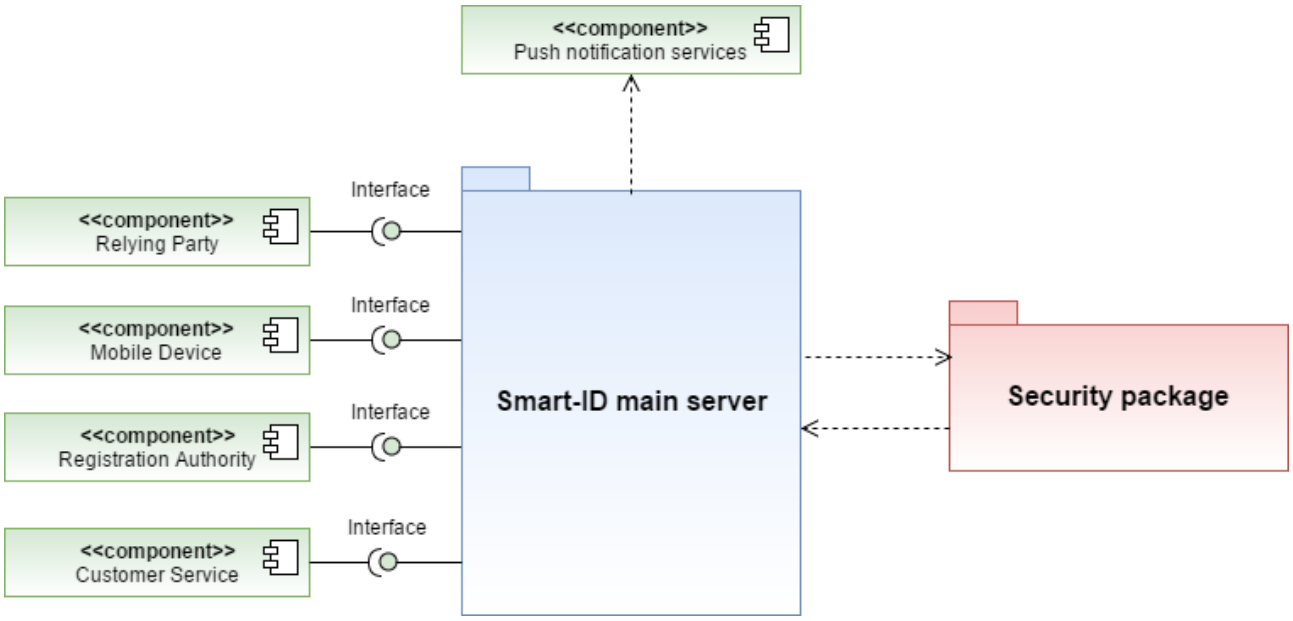

Figure 1 - Simplified Smart-ID internal architecture and relationships

The application needs to be thoroughly tested in order to determine whether it complies with client's functional and non-functional requirements. To perform testing we will take the Gatling testing framework as a basis and extend it to the point it will comply with the requirements for the application testing solution.

3.2 Description

This section will give an overview of the application we used for testing in the particular project. It will describe product perspective and functions, software expectations, user characteristics, constraints and requirements.

3.2.1 Product Perspective

The product is a testing application based on the Gatling stress testing framework (2.1.4), which main functionalities are:

- Smart-ID backend server API calls
- test scenarios creation
- test scenarios single execution
- test scenarios parallel execution

- test data preparation
- database operations (e.g. read/insert data)
- test report generation
- log generation

Some of the mentioned functions are already integrated in the Gatling framework, such us different test scenarios operations and report generation. All other required functionality (e.g. complex API calls) has to be implemented manually as Gatling only provides basic HTTP client functionality. After the required support has been added, one can start with constructing test simulations[11] and scenarios[12].

The application requests destination is the main server (*Figure 1*). Thus, the other system components (mobile device, relying party, registration authority and customer service portal calls) must be emulated by the application. The detailed testing application's internal structure and external entities relationships can be seen in the *Figure 2*.

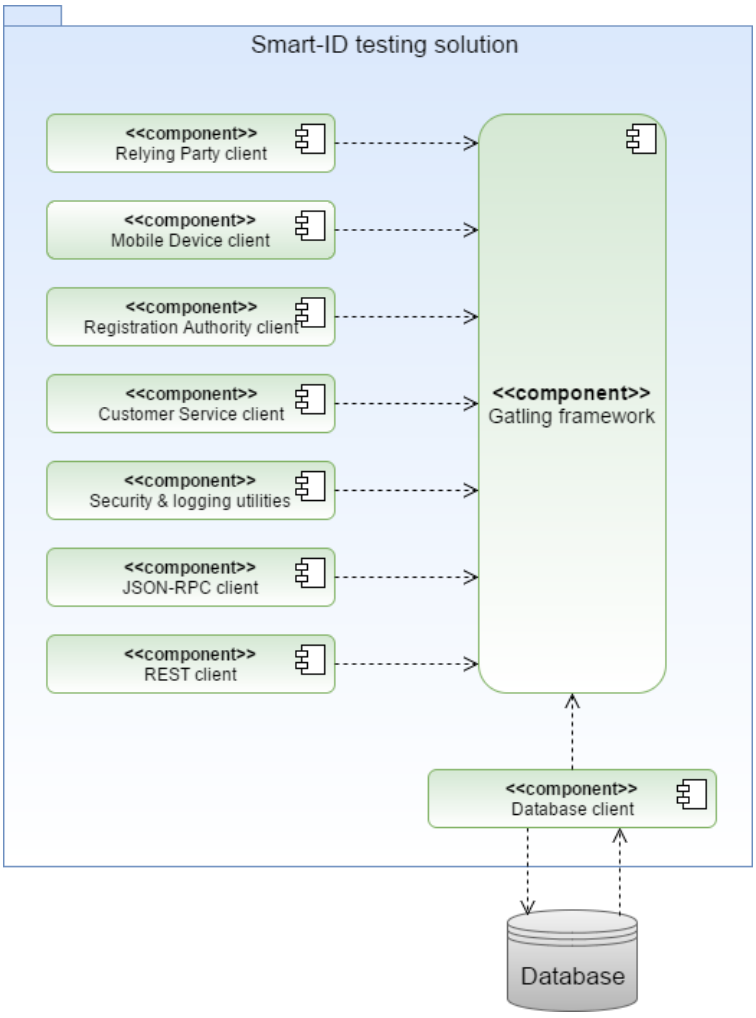

Figure 2 - Detailed testing solution's internal architecture

[11] A simulation describes how, possibly several, user populations will run: which scenario they will execute and how new virtual users will be injected [8].

[12] A scenario represents a typical user behavior. It's a workflow that virtual users will follow [8].

The main users of the application must have programming experience, because tests are programmed as scripts. However, the Gatling DSL is present, which makes the programming knowledge requirement secondary.

The application has support for all the Smart-ID application API calls. Therefore, the user must define which request should be sent. The following request parameters are considered:

- URL
- body
- authentication[13]

All other required parameters, such as request headers and request type, are set automatically.

3.2.2 Product Functions

Create simulation

The tester creates a new simulation, which describes what scenario/scenarios will be running. The tester also defines the simulation parameters (e.g. immediate users amount).

Create scenario

The tester creates a new test scenario, which describes what tests will be executed during the scenario simulation.

Create test

The tester creates a new test which describes what requests will be sent. The tester must choose either predefined/default requests or provide request bodies and expected output (request response).

Prepare test data

The database is truncated and test data from predefined SQL script is inserted into the database by the tester.

Run simulation

The tester starts the defined simulation and observes the execution in real-time.

View simulation results

The simulation results are generated in the specified folder and can be analysed by the tester.

13 HTTP Basic authentication.

Product expectations

The following table describes what are expected conditions so that the product functionality could be put in use. These conditions include the user having the application distribution and programming experience as well as the product having access to the application's database and server.

Table 1 - Product expectations

ID	Statement	Description
PE-1	User has the application distribution	In order to use the application the user must have access to its distribution
PE-2	User has programming experience	In order to add new simulations, scenarios and tests the user must script them
PE-3	Product has access to application database	In order to interact with database the product needs the database credentials configured
PE-4	Product has access to application server	In order to send requests to the application server the product needs server parameters configured

3.2.3 Software Expectations

The following table describes the environment conditions that are required to run the software. These conditions include the application being deployed, configured, running, and having access to database and ActiveMQ JMS service.

Table 2 - Software expectations

ID	Statement	Description
SE-1	Application is deployed and running	In order to respond to requests the application must be built, deployed and started
SE-2	Application has access to a database	Database must be running in order to interact with the application
SE-3	Application has access to ActiveMQ	ActiveMQ must be running so that application could send and receive messages through its JMS implementation
SE-4	Application is configured	In order to function the application configuration must be set up in the database

3.2.4 User Characteristics

The test application has 2 main user roles:

1. Software developer – the developer's main goal is to adapt the application to a particular project by implementing required functionality, such as different API calls and messages, utilities, configurations. In addition to it, a developer can also write and execute tests, as well as analyse the results.
2. Software tester – the tester's main goal is to use the application to add new simulations, scenarios and tests, as well as execute them and observe the test reports.

Both roles are expected to have programming experience to a particular extent. Additional features implementation requires more knowledge (Scala programming language) than constructing test simulations using Gatling DSL.

3.2.5 Constraints

Since the main goal of any testing software is to assure the final product's quality, some important metrics are needed to be taken into consideration during the testing process:

- memory pools utilization
- request response times
- connection pools (e.g. database, JMS)
- active threads

These constraints can be overcome by using certain system monitoring techniques and tools[14].

3.3 Requirements

3.3.1 Protocol Support

System modules use 2 different protocols for communication. Both protocols' support must be implemented in the testing application in order to be able to send requests to the server and receive positive responses.

Table 3 - Protocol support

Protocol name	Description	Specifics
RESTful API	Representational state transfer. It relies on a stateless, client-server, cacheable communications protocol -- and in virtually all cases, the HTTP protocol is used.	RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations [14].
JSON-RPC 2.0	JSON-RPC is a stateless, light-weight remote procedure calls (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing.	It is transport agnostic in that the concepts can be used within the same process, over sockets, over http, or in many various message passing environments. It uses JSON as data format [15].

[14] This aspect is covered in the chapter 2.4.

3.3.2 External Interface Requirements

The following table describes the products external interface support requirements in order to perform its basic operations. These interfaces are Oracle database and Logback logging framework.

Table 4 - External interface requirements

Interface	Description	Specifics
Oracle database	The system uses Oracle database in order to persist the data. The test application must be able to access the data to perform basic operations/checks.	A database client must be implemented in the application using Scala. It must support 4 CRUD operations.
Logback framework	The test application must be able to use Logback framework to do all the logging during the simulation execution.	The test application must log all outgoing requests, incoming responses, database operations, session variables.

3.3.3 Functional Requirements

Table 5 - Functional requirements describes what functionality must be implemented in the testing solution in order to perform basic operations. The framework must support calls to mobile devices, relying parties, registration authorities and customer server portals. Additionally, it must be capable of test data preparation and insertion into database tables, test report generation, concurrent test execution and basic operations' logging.

Table 5 - Functional requirements

ID	Statement	Description
FR-1	Mobile device API calls	The application must be able to send requests to mobile device server API and receive responses.
FR-2	Relying party API calls	The application must be able to send requests to relying party API and receive responses.
FR-3	Registration authority API calls	The application must be able to send requests to registration authority API and receive responses.
FR-4	Customer service API calls	The application must be able to send requests to customer service API and receive responses.
FR-5	Database test data preparation	The application must be able to take SQL scripts as input and execute the provided scripts to the given database tables.
FR-6	Test report generation	The application must be able to generate a final report in the end of each test simulation with the following statistics: number and percentage of failed /succeeded requests, minimum / maximum / average response times.
FR-7	Logging	The application must be able to log each incoming and outgoing request's header and body objects.
FR-8	Test parallel execution	The application must be able to execute tests concurrently.

3.3.4 Non-functional Requirements

The following table describes the testing solution's non-functional requirements that have to be fulfilled in order to increase its quality. These requirements cover maintainability, portability, performance, adaptability, stability, usefulness and documentation aspects.

Table 6 - Non-functional requirements

ID	Statement	Description
NFR-1	Maintainability	The application must be easy to maintain.
NFR-2	Portability	The application must be portable to our systems at least partly (the most used functions).
NFR-3	Performance	The application must be able to generate load while not stressing the machine it is being executed on with the exception of operations, which require much computing power.
NFR-4	Adaptability	The application must be adaptable in case of target API changes.
NFR-5	Stability	The application must be stable during the test simulation executions.
NFR-5	Usefulness	The application must meet relevant needs as regression and stress / load testing tool.
NFR-6	Documentation	The application must be self-explanatory to use or contain sufficient descriptive information.
NFR-7	Source code	All source files must be in English.

3.4 Summary

In this chapter we provided the tested application's structure. Additionally, we have outlined the testing framework's structure and usage patterns (functions, users, pre-conditions). Finally, we described the adaptation process by indicating the required protocols support, which had to be implemented, as well as the functional and non-functional requirements, which had to be complied with.

In the next chapter we are going to provide the results of using the selected testing framework for QA in the mentioned project. We will discuss the encountered problems and provide possible solutions.

4 Test Process Execution

In this chapter we will present and discuss our approach to the test processes execution. We will first debate on influence of the application testing on its architecture (4.1). Then we will take on the testing design in details (4.2) and the testing implementation outcome / lessons learned (4.3). We will finally discuss on how one must approach the task of test and test environment design construction (4.4).

4.1 Application Testability

Application testability stands for testing the maximum possible percentage of application codebase with the maximum possible amount of input with the minimum resources. In simple terms it could be defined as a system characteristic of how easy it can be tested. The degree of ease must take the following aspects into account:

- time used to design and implement test code
- required professional skill to extend a used testing framework to cover all important application aspects

The higher level testing is needed, the more expensive the tests are in terms of the aforementioned resources. Testing an application on functional and object level with unit tests is usually trivial and cheap. Injecting as much dependencies as possible into unit tests increases their overall usefulness and functionality coverage. Although unit tests do not guarantee that the application is fail-proof, functional nor stable, they provide immediate feedback, which allows to make the changes to code faster. Faster feedback is achieved with the system components simulation or dependency injection such as:

- In-memory database with automated database schema creation – it is faster than applying a database schema to a standalone instance and clearing it up after each test.
- Simulated network – unit tests have to operate offline and provide fast feedback, thus, proper design avoids using network protocols and simulates application internal calls.
- Simulated time – there is no point in waiting for a timeout if we can simulate it and move on with following requests.
- External interfaces – as network protocols are avoided and the speed is the number one factor it is also important to simulate external interfaces, or in other terms write mock implementations. It is not as reliable as testing against a real interface, but it is not the external interface that we are testing. Our main goal is to receive feedback of how our application units are functioning and whether the output is as expected.

In addition to speed prioritization it is also critical to avoid non-determinisms in the test design such as multi-threading or random generators. Non-determinism, in terms of software testing, can be defined as non-predictable outcome of unit tests, which is completely against the goal of unit testing – ensuring the predicted output is received.

What is more important, writing unit tests beforehand, also known as Test Driven Development (TDD), forces to provide clear design of the components to the application with each separable task in mind. Unit tests assist with designing the application code – instead of writing the implementation first, they force a developer to outline all code conditions, possible input and expected output.

Application automated test suites take focus on different aspects, such as providing overall picture of the application functionality and stability as well as feedback on its interfaces separately. As opposed to the unit tests, automated regression tests do not use application code directly, thus the different codebase is needed, which requires additional code design. As a result, automated tests are more expensive compared to unit tests. The automated test code has to follow the same quality requirements

as the application code itself. With the application complexity growth, it becomes even more important (4.3).

A specialized test framework, running the automated test suite, is a completely different project apart from the application and does not use its resources directly. Therefore, the whole ecosystem has to be configured to accept test requests:

- Standalone database with applied application database schema – a test database schema has to be updated along with an application server schema update.
- Test data – keys, configurations, personal data has to be inserted before test execution.
- Java messaging queues (JMS) – JMS framework needs to be configured so that a test framework could send messages to queues and listen from them.
- Periodical workers – these are responsible for clean-up and resubmission threads.
- External interfaces – they have to be configured in order to perform basic operations and workflows with a test framework. In our particular project, these include certificate and relying party authorities, HSM[15].

Automated tests force to design application interfaces architecture with different aspects in mind, in contrast to unit tests. First of all, the process of automated test execution need to be simplified as much as possible, which involves automatic test environment setup from scratch (4.4). In addition to it, an application must be flexible to increase its testability, which is a major factor for automated test execution. In simpler terms, this means that the most of application variables (e.g. JMS queue names, timeouts, keys, etc.) have to be made configurable on-the-fly. Finally, to construct proper automated tests, the application behaviour needs to be considered on API level which means designing the communication between modules and describing each module possible request along with the output with the values, which need to be persisted, taken into account. All in all, the application must be designed with testability taken into consideration in order to increase the quality of both the application and the test code.

4.2 Tested Application Architecture and Test Design Cross-influence

We have previously briefly introduced the definition of application testability. We have focused mainly on unit tests and automated tests design processes as well as how the different test construction forces to provide better application design, when the tests are taken into consideration beforehand. In reality, this is a double-sided connection. This means that the application, taken its concrete and specific design, can as well assist the test process execution.

Taking our specific project as an example, the automated test suite does contribute to providing all application modules quality assurance upon each application code update. With the current state, the modified Gatling testing framework executes overall more than 2000 requests within roughly 10 minutes. These numbers have grown over past year as the application complexity and functionality evolved. All the requests are executed sequentially, which has become a problem with the current test suite size. However, Gatling is asynchronous, which allows to send concurrent requests. Given our Spring application's natural parallelism, the tests in the test suite can be redesigned to be able to run concurrently, which will decrease the complete suite execution times, depending on the execution times of tests being run. In theory, when all tests in the suite are executed concurrently, the entire test suite execution duration decreases to the longest test of the suite.

[15] Hardware security module

This step requires additional time and resources allocation. However, it would benefit on the later stages of development as the application codebase continues to grow. The main redesign problem would be test data usage as all the tests use the same predefined data which is inserted before test execution. One possible solution would be to use an application itself to provide test data dynamically on the fly and holding key session data in a test framework's memory. This way automated tests would benefit from different aspects:

- Avoid race conditions when 2 or more tests are accessing the same data.
- Test data maintenance is no more needed.
- Faster feedback is provided due to test execution concurrency.

Let us return to a topic of non-deterministic design of system components. How does this problem actually influence the test design? The main idea is to develop a deterministic approach to testing non-deterministic system components. We can take the account registration module as the example - specifically - the asynchronous CSR (Certificate signing request) resubmission worker. In order to complete the registration process, an account is obliged to receive valid certificates from an authorized certificate authority (CA). If such authority is not available at the moment of registration, then it falls into the *pending* state and the CA request has to be repeated periodically. This operation is autonomous and cannot be triggered manually, which makes it difficult to test.

One of the possible solutions would be to design and implement additional APIs on an application server that would be specific to test executions. In our particular case, we implemented triggers for asynchronous workers, that allow us to activate them on-demand and check the request results on-the-fly.

Another solution would be to design specific tests using the test framework. However, one should refrain from the common "send request - check response" logic and take an application's behaviour and entities' statuses into consideration. Returning to our registration process example, we can design a test, which would limit communication between the registration module and CA. As a result, we should receive several retry requests (depending on application configuration) and registration status changing from *pending* in the test beginning to *complete* in the end.

4.3 Test Design and Test Processes Execution Outcome

Our application regression testing process fully relies on Gatling test simulations. The initial thought was that Gatling own DSL would simplify the test design and construction processes by making the programming knowledge requirement secondary. However, it stands true only in case if no additional extensions to the framework are required. In other words, if a framework's default functionality package does not satisfy testing needs, then the framework requires additional extensions. As test and application code quality have to be on par with each other, test framework extensions development relies on a developer's time as well as programming knowledge.

It is also important to notice that during test design and scripting processes one must take into consideration the constantly growing and significantly changing nature of some of an application's APIs. Poorly written tests eventually require refactoring, which is time consuming, depending on the test codebase size. But how can we deal with the problem of constantly evolving application API during the test design?

First let us compare two simple functions A and B from an application API. The function A requires 2 String type arguments while the function B requires only 1 numeric argument. The simplest approach would be to write 2 separate custom test functions - a test function for each API function. But what happens if we need to add another API function to a test code? We implement another custom test function manually. What if the function A requirements' list changes? We refactor the

custom test function we have previously written. But what if there are tens or hundreds of functions, which are called from each other, and they are constantly changing? In this case, the test code maintenance becomes a serious problem.

The solution to this problem is automation. It is important to automate every single part of the test framework which can be automated. First of all, the input given to test functions must be minimal in order to provide the output (this concerns the application design as well). Returning to our example, the most logical solution would be to write one general test function for all similar API functions which by the number of the variables and their type would automatically refer to the required resource. This solves the main problem of designing tests for constantly evolving application – future API changes would require minimum additional test functions implementation / refactoring. As a result, the number of test functions is decreased and the test code is better structured. Thus, the code maintenance would consume less time.

4.4 Approach to Test Design and Environment Construction

A tested application must first of all be designed. The design process must be thoroughly documented and a resulting document well-structured. In case of an API design document (system description) well-structured means that the document is split into straightforward, clear and understandable pieces. This results in a better test quality, as the documentation pieces can also be applied to the test design process. But what are those document pieces and based on which criteria a document should be split?

It is very important that a document structure corresponds to an application structure and a test structure and the other way around. The major pieces which are important to differentiate between:

- Modules - major application parts that can be deployed separately if necessary.
- Controllers - the module interfaces that are responsible for taking input and providing output.
- Services / components - the parts that provide the main functionality which is used by the controllers.
- Utility functions - single functions that are required by the application services.

The differentiation between these pieces allows to structure application and test code appropriately. It also assists in understanding how different modules communicate to each other - both the application and the test framework. This integrity guarantees overall code quality - it becomes easy to differentiate between different system pieces and debug them.

Another important aspect that should follow concepts of integrity is source versioning. Usually project sources are a combination of the following components:

- Source code
    - Application code
    - Test code
    - Different utilities and setup scripts
- Documentation
    - Application overview, structure and API documentation
    - Installation and configuration guide

Whether project source code and documentation are stored together (in the same repository) or not, it is important to properly version them in the way that their versions conform to each other. In our company’s ecosystem documentation and source code are stored separately in 2 different repositories. When a new software version is to be released the latest states of the repositories are checked out to

a new release branch which is then forwarded to a customer. This preserves the application sources integrity and provides access to all released versions with corresponding documentation and source code in case of need (version specific software bugs and fixes).

Tests have to be designed against a particular test environment, preferably the same that is used for the tested application. This guarantees test process relevance - the used test environment is not abstract and tests actually output the same results as the production system would. However, as application grows and becomes more complex - additional dependencies appear - environment also evolves. What can be done in order to preserve test execution and environment setup simplicity?

As it was mentioned previously, everything that can be should be automated. This concept concerns test environment setup process as well. In our case test environment setup process includes the following components configuration:

1. application server - Tomcat
2. database node - Oracle
3. database schema and test data - a list of SQL commands
4. application modules - a set of .war containers

Manual setup process is time consuming and should be automated in order to save resource in long-term[16]. This is another example of a long-term resource investment in development procedures - environment setup automation drastically decreases the time spent on configuration and maintenance as well as the chance of causing errors during a setup process.

4.5 Summary

In this chapter we covered such topics as application architecture design, test framework and tests design, automated environment setup, documentation. We have discussed the importance of these topics and the problems we encountered, and how they were solved. In the next chapter we are going to validate the outcome against the experts' opinion in their particular fields.

[16] Chef is one of such frameworks which allow to build, deploy and manage infrastructure and applications fast, efficiently, and with far less risk compared to manual management (learn more).

5 Validation

In the previous chapter we pointed out the key points of the following processes / topics:

1. Test framework design.
2. Test environment setup.
3. Unit and regression test design.
4. Application design with testability in mind.

In this chapter we are going to validate the before mentioned key points against the experts' opinion in their mastered fields:

- software engineering
- system architecture design
- software quality engineering

The validation is going to be conducted by a series of head-to-head interviews in which the participants are presented with the interview program, the outline of the key points and the recommended practices. The participants are encouraged to provide their personal expert opinions on the topic as well as previous experiences.

5.1 Interview Program

The main point of the interview is to validate the provided development key points and recommendations in terms of usability which is usually a combination of 2 things [16]:

- Fit of use – Are the provided claims useful and do they cover important aspects of tests and application architecture design.
- Ease of use – What amount of time, programming knowledge and resources does the implementation of the provided claims approximately take and how efficient are they in the day-to-day work.

When referring to the ease of use aspect of the interview it is important to address the following topics:

1. Ease of learning – How easy it is to learn how to apply the provided recommendations in a particular project for various users' groups.
2. Task efficiency – How efficient the provided claims are to a frequent user.
3. Ease of remembering – How easy it is to remember the instructions.
4. Subjective satisfaction – How satisfying it is to follow the recommendations.
5. Understandability – How easy it is to understand the recommendations on a deeper level (why it is important to follow them and how they affect a target system).

The task of the interviewed expert is to provide his/her opinion on the recommendations and guidelines provided in the previous chapter with usability components[17] taken in mind. The interviewee is also encouraged to provide additional feedback:

- Do You plan to / already use the claims mentioned?
- Is there other important topic that is important to be take into consideration, but is not covered?

[17] Fit of use and ease of use definitions are provided in 5.1.

5.2 Interviews

Software engineer, Informatics undergraduate

Fit of use:

The recommendations provided are useful and definitely are worth considering. All important aspects are covered. The only thing that comes to mind is that when discussing the application and test design sometimes the decisions are based on whether there are existing solutions and utilities that can be used to fulfil the requirements. In this way we decrease the resources needed for the tool adaption. However, such decisions are project specific.

Ease of use:

It is easy to understand the concepts and the reasons for implementation. The implementation itself, on the other hand, is a more complex and resources demanding activity. The major thing to consider is a development speed against the quality (different recommendations aim to improve different metrics). Again, the decisions must be made according to the specific project requirements.

The instructions are easy to remember to follow, unless they are completely ignored. In other words, the activity of complying with the recommendations has to become a development process routine. The decision depends on whether the development team considers this activity valuable.

Generally, following the recommendations bring a great degree of personal satisfaction. First of all, the development process becomes more complex, but brings a feeling of completion – important project parts are automated (configuration, setup, testing, etc.). Furthermore, some development aspects, as testing and environment setup, become verifiable – it is possible to run a script and receive an immediate result.

As mentioned before, the complying with the recommendations is a project specific decision. In order to understand the necessity of following the provided guidelines, it is important to analyse a concrete project's functional and non-functional requirements.

Software developer, Informatics bachelor's degree

Fit of use:

The recommendations do cover the targeted scope. The results, which in theory can be achieved by following the mentioned practices should increase overall product's quality and in a company that develops critical systems it is important. One of the topics, which could also be taken into consideration is use case prioritization. Usually, it is important to target a system's critical functionality first when designing automated tests. The critical functionality is often derived from a use case document, which is created by a collaboration between an analytic and a client. Another thing, that could be kept in mind, is that it might be too complicated to write automated tests for some specific functionality and manual testing could be used instead. But again, this is a specific decision, based on a project's goal, team size and used technologies.

Ease of use:

All recommendations provide useful outcome when used (less errors, time saved on routine tasks, etc.). However, some of the are not as easy to learn to use as the other, e.g. unit testing versus application testability. The former is a trivial process, the latter requires more analytical and overall development knowledge.

It is easy to remember to follow the recommendations. However, it must be done routinely. Otherwise, they will be neglected.

Task efficiency is a complex topic. Overall, proper application testing and processes' automation certainly increases efficiency if by efficiency we mean the speed of development. One might think that focus on quality could slow it down, but providing that systems have bugs, we potentially save time, which would be spent on refactoring code and fixing errors.

Important processes automation saves time from doing the routine work iteratively and automated tests bring a verification factor to an application development (test can always be run and check whether there are no critical errors). These are preconditions for a personal satisfaction, and it can be guaranteed by following the recommendations.

It is easy to understand why the provided guidelines should be followed, as they are not particularly complex. Overall the recommendations are important to follow and provide improvements[18] . However, they are often not taken into consideration, as it is a project specific decision, especially in case of large companies.

QA engineer, Computer science master's degree

Fit of use:

In general, the most of the recommendations mentioned are important to follow as they help to receive immediate feedback and improve a product quality. However, some aspects are covered briefly – test concurrency and system state preparation is a critical topic. Some of the recommendations could not be applied because test data preparation could be a very resource (CPU, memory, time) demanding operation, which should not be repeated for each separate test instance. Overall, the provided guidelines are project specific, as different systems' input and output may vary. The recommendations are focused on one concrete example and sometimes do not provide a good insight into a bigger picture.

Ease of use:

Obviously, depending on a previous experience, it might be hard to learn to use the recommendations for the first time. Another important point, is that if a person is trained to follow completely different practices, it will be even more difficult to apply the given recommendations. Nevertheless, most of the recommendations are basic and should be ease to learn.

Task efficiency depends greatly on a system size. It is usually low in the beginning of a project and grows high with the project complexity and codebase. The usage of the given practices should be referred to as a long-term resources investment.

It is actually easy to remember to use the recommendations if they were applied in terms of a concrete project and worked well.

Having a highly testable API and an ability to quickly find critical problems within a system definitely bring satisfaction. Moreover, when taking documentation into consideration, reading a well written and structured document is always a pleasant experience.

The recommendations are easy to understand to some degree. However, without a context or more specific examples it is not as trivial and some of them may seem simplified. Again, it is a project specific issue.

[18] Mentioned before, e.g. code quality improvements, increase in development speed, time saved.

Software developer, Informatics master's degree

Fit of use:

Generally, the recommendations are important to follow. Moreover, in the majority of projects some of them (e.g. documentation and source code integrity) are considered essential and are thoroughly controlled. Most of the recommendations are only applicable in case of a large-scale projects, as the scope says. These guidelines are not suitable for smaller projects. In this case suitability means that long-term resource investments (for example, scripting automatic environment setup) are not profitable due to the shorter nature of such projects. Finally, some examples, provided in the text, were not clear and require more context.

Ease of use:

It is easy to learn how to apply the presented recommendations. However, the implementation process itself can be difficult and time demanding, depending on for how long the project has been going. Usually, the more complex the project is, the harder it is to change the application and tests on design and architecture levels.

If our main goal is assuring an application's quality, then the recommendations increase the work efficiency. However, if we focus on the development process and providing new implementations as fast as possible then the provided guideline can slow the processes down.

If the person is motivated to follow the recommendations, then it will be easy to remember them. The claims need to be forced to be used and become a routine in order to be taken into account.

Automated processes in software development always provide a high degree of satisfaction as they aim at removing time-consuming routine jobs (e.g. environment setup / configuration, test data generation, etc.) and enable developers to spend their time on more important issues.

It is easy to understand why the claims provided are important for an experienced user. Without meaningful experience (1-2 projects depending on size and duration) the recommendations may seem hard to comprehend.

5.3 Bottom Line

The table below describes the results of the conducted interviews – all fields refer to the opinions on the presented recommendations. Answers have been simplified and grouped by fit and ease of use factors. The latter is a combination of 5 topics: ease of learning, task efficiency, ease of remembering, subjective satisfaction and understandability.

Table 7 - Interview results

	Fit of use	**Ease of learning**	**Task efficiency**	**Ease of remembering**	**Subjective satisfaction**	**Understandability**
1	useful; all important aspects are covered	easy concepts; implementation is complex	decrease speed of development; increase product quality	easy if not ignored	completion due to verification provides satisfaction	depends on a specificity of a project; requires analysis
2	cover the scope; some claims are specific to a project	complexity varies (unit testing - low complexity versus testability – high complexity)	automation increases efficiency	easy if followed routinely	recommendations provide verification which brings satisfaction	recommendations are not complex
3	help to receive feedback; improve product quality; some aspects are covered briefly; context specific	hard for first time; easy and basic for experienced user	grows with system size	easy if were already applied in a project	highly testable API and well written documentation provide satisfaction	easy to understand to some degree; are not trivial without context
4	important to follow; project specific; project size is an important factor; long-term investment	easy to learn basics; the implementation itself is complex; depends on project	increase product quality; decrease implementation speed in some cases	easy to remember if there is motivation	automation provides satisfaction	comes with experience

All of the interviewees agreed that the provided recommendations mostly cover the targeted scope as well as the most important application and test design aspects, and should be taken into consideration. Despite the fact, that these guidelines were described using a concrete project as an example, it is not safe to assume that they can be applied and provide the same outcome when used in other projects. This decision is project specific and heavily relies on a client, who is providing resources for a product development. This also explains why the recommendations do not cover all possible aspects and might be simplified in some cases.

It is also clear that the recommendations are not complex and can be learned to apply easily (depending on a previous experience). However, the implementation process can be complicated and resource demanding, and should be treated as a long-term investment. In order to remember to follow the provided guidelines, it is important to make the considered activity a routine and do not neglect it. A personal satisfaction factor is also important, as following the provided practices gives a feeling of completion, saves time, removes iterative activities and brings a certain degree of verification to application development process. Finally, it is easy to understand why the recommendations should be followed, but again, when taking projects' diversity and specificity into account, it might be needed to conduct further analysis[19].

5.4 Threats to Validity

We identified that one of the possible threats to validity is the specificity of the resulted set of recommendations. Although they derived from an enterprise grade Spring application and should apply to similar systems, it might be the case that due to some particular project specificity[20] it would be impossible.

The another threat could be the insufficient feedback collected. Although each of the experts had a different real-life experience within software development processes and the participants were interviewed excessively[21], the total amount of 4 reviews may seem lacking. However, due to the interview program topics, the systematic approach to the feedback collection is problematic as well as time consuming.

5.5 Summary

In this chapter we described the technique we used for validating our test processes execution outcome – a series of interviews with experts in their particular fields. Moreover, we presented the interview program as well as the interview results. Finally, we have summarised the interview results and outlined the final verdict.

In the next chapter we will discuss the work completed, as well as the encountered problems and limitations. Finally, we are going to summarise the thesis and provide an insight on what future work might theoretically be completed to improve the results.

[19] One of such documents, which provide an insight into what practices must be applied, is a project's functional and non-functional requirements document.

[20] In the interview with QA engineer it was mentioned that in some cases test data generation is an extremely demanding process, and running it multiple times should be avoided at all costs. Again, it was a system specific claim.

[21] Each discussion on fit / ease of use topics lasted 2 hours on average.

6 Conclusion

In this paper we introduced a number of stress testing frameworks – Locust, Apache JMeter and Gatling project. One of our goals was to understand which of the selected frameworks suits best for our particular project. We have completed this task by describing the basic principles of load and stress testing and provided selection criteria based on which a comparison was made and the tool was selected. We also briefly discussed the selected testing frameworks limitations and how they can be compensated by certain system monitoring techniques. Furthermore, we have provided a tested application background, as well as particular requirements and pre-conditions set to the testing framework. We have also showed the framework extension process by describing the protocols support and functional / non-functional requirements, which had to be complied with. Finally, we have described the test execution process, outlined the results and validated them against the experts' opinion through a series of head-to-head interviews. We conclude our study by discussing the limitation factors and answering our main research questions.

During our research we have encountered a few limitations, which have had affected our results. First of all, the testing framework has been selected based on the project specifications, requirements and our personal preference. The selection was made of 3 particular frameworks. However, there are much more available tools which were out of our scope. Additionally, the final set of recommendations was based on our current project. As a result, they are not applicable to every other project with its own specifics (e.g. resource demanding test data generation). Lastly, the collected feedback might have been insufficient. Although the interviews were as thorough and complete as possible, we have collected reviews from only 4 experts, which may be not enough to form a final opinion.

As a result, we have selected the Gatling project as our test automation framework for the following reasons:

- Test scenarios are readable, maintainable and can be handled by the version control system (Git).
- It is supported by the continuous delivery tool (Jenkins) through the plugin.
- Gatling tests can be run through Gradle build tool.
- It provides real-time test execution monitoring and report generation.

Overall, the Gatling occurred to be a highly configurable platform and complied with the client's non-functional requirements, such as running on a JVM and having a good performance (generating high load on a target server while not stressing the injector machine where it is being run).

During the test processes execution within the project our team has encountered a series of problems such as slow test execution, low maintainability of the test code, insufficient application testability and more. As a result, we concluded a set of considerations[22] for application and test design:

- Tests have to provide as fast feedback as possible. There are certain techniques to achieve that[23]. Tests' and test framework code quality is an important issue. Test functions should take the least possible amount of input to be maintainable. Test code should avoid non-determinisms.

[22] The listed recommendations derive from more specific examples and solutions presented in the chapter 4.

[23] In case of unit tests these are, for example, using in-memory database and simulating network, time, external interfaces.

- Writing (or at least designing) tests before writing implementations helps to create better application structure.
- Application concurrency, configurability and well written / structured documentation help with testing when tests are designed properly.
- Application non-determinisms can be tested using a certain system design and testing approach. In case of asynchronous processes, the particular triggers can be implemented in order to allow calls within test functions. It is also possible to design tests with non-deterministic approach.
- Automatic test and development environment setup saves a lot of worktime in long term, especially in case of complex projects.
- Documentation plus source code structure and versioning should conform to each other. This results in a more efficient development, testing and debugging activities.

Considering the limitations our team has encountered, the results can be improved. As future work, we, first of all, must overcome the following difficulties:

- Recommendations specificity – the current set of guidelines derives from the particular project experience. We must try to follow them during the other projects as well to see how well they comply with different functional / non-functional requirements and clients' needs.
- Feedback collection – the interview format presented in the chapter 5 limited us with the amount of information we could collect. It would be possible to collect the feedback from a wider audience if the format is changed to an online survey type. It would allow to collect more information in a faster way. However, when designing the survey, questions require exquisite attention to detail.

We believe that completing these steps will improve the results presented in this paper as well as the validity of the research.

7 References

[1] C. Walls, Spring in Action, Fourth Edition, NY: Manning Publications Co., 2014.

[2] Pivotal Software, "Spring Framework Reference Documentation," 2017. [Online]. Available: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/.

[3] J. Heyman, C. Byström, J. Hamrén and H. Heyman, "Locust," 2017. [Online]. Available: http://locust.io/.

[4] A. Jones, "Load Testing with Locust," 15 August 2014. [Online]. Available: https://andrew-jones.com/blog/load-testing-with-locust/.

[5] D. Ingram, "Load Testing with Locust," Promptworks, 2 June 2016. [Online]. Available: https://www.promptworks.com/blog/load-testing-with-locust.

[6] Apache Software Foundation, "Apache JMeter™," 2016. [Online]. Available: http://jmeter.apache.org/.

[7] E. H. Halili, Apache JMeter, Birmingham: Packt Publishing Ltd., 2008.

[8] Gatling Corp, "Gatling Project, Stress Tool," 2016. [Online]. Available: http://gatling.io/#/.

[9] R. Tolledo, "Gatling: Take Your Performance Tests to the Next Level," 12 May 2014. [Online]. Available: https://www.thoughtworks.com/insights/blog/gatling-take-your-performance-tests-next-level.

[10] B. Wescott, Every Computer Performance Book, CreateSpace Independent Publishing Platform, 2013.

[11] I. Molyneaux, The Art of Application Performance Testing, Sebastopol: O'Reilly Media, 2014.

[12] A. Reitbauer, K. Enzenhofer, A. Grabner and M. Kopp, "Analyzing the Performance impact of Memory Utilization and Garbage Collection," in *Java Enterprise Performance*, Dynatrace.

[13] Oracle, "Oracle9i Database Performance Tuning Guide and Reference," 2000, 2002. [Online]. Available: https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#g42231.

[14] D. M. Elkstein, "Learn REST: A Tutorial," February 2008. [Online]. Available: http://rest.elkstein.org/2008/02/what-is-rest.html.

[15] JSON-RPC Working Group, "JSON-RPC 2.0 Specification," 4 January 2013. [Online]. Available: http://www.jsonrpc.org/specification.

[16] S. Lauesen, in *Software Requirements: Styles and Techniques*, Pearson Education Limited, 2002.

[17] S. &. K. Penchikala, "Software Testing With Spring Framework," 12 November 2007. [Online]. Available: https://www.infoq.com/articles/testing-in-spring.

[18] M. Wacker, "Just Say No to More End-to-End Tests," Google, April 2015. [Online]. Available: https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html.

Appendix

I. Glossary

Spring Framework Application framework, which supports dependency injection, built for Java platform..
Locust Open-source stress testing framework written in Python.
Apache JMeter Open-source Java application used to conduct systems' load, stress and performance testing.
Gatling Open-source stress testing framework written in Scala.
Java, Scala, Python Object-oriented computer programming languages.
JavaScript High-level, dynamic, untyped, and interpreted programming language.
JSON JavaScript Object Notation, a format of data.
Microsoft Silverlight Application framework designed for .NET platform.
Akka Framework written in Scala, designed to build concurrent and distributed systems .
Netty Network application framework.
JVM Java virtual machine, which runs Java programs.
MVC Model–View–Controller, a pattern of software design.
REST Representational state transfer, state information management architecture.
API Application programming interface.
JDBC Java Database Connectivity, Java API, designed to provide database access.

JPA Java Persistence API, relational data management specification on Java platform.
JMS Java Message Service, Java service, designed to send messages.
J2EE Java Platform, Enterprise Edition / Java EE, enterprise grade systems' programming platform.
(G)UI (Graphical) user interface.
HTTP Hypertext Transfer Protocol, standard protocol, which enables to distribute information over the web.
HTTPS HTTP protocol extension designed to improve security.
SOAP Simple Object Access Protocol, message exchange protocol.
FTP File Transfer Protocol, standard client-server network protocol.
DSL Domain-specific language, system specific programming language.
CPU Central processing unit, a piece of electronics executing machine instructions.
SQL Structured Query Language, DSL designed for relational databases.
KPI Key performance indicator.
Maven Build manager designed for Java based projects
Gradle Open-source tool design to automate application build process.
Garbage Collection (GC) Form of automatic memory management.
Young generation Describes GC, newly created objects, consists of 1 Eden space and 2 Survivor spaces.

Old generation Describes GC, objects that survived from the young generation.
Coroutine Application component, which provides flexibility (multiple entry points, execution pause and resume).
Greenlet Package, written in Python, designed to support concurrency by adding micro-threads.
Libev Package, event loop.

Appendix 1 - Used terms glossary

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Andrei Proskurin,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:
   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Adapting a Stress Testing Framework to a Multi-module Security-oriented Spring Application,

(title of thesis)

supervised by Mart Oruaas and Raimundas Matulevicius,

(supervisor's name)

2. I am aware of the fact that the author retains these rights.
3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 18.05.2017