



LAUREA
UNIVERSITY OF APPLIED SCIENCES
Together we are stronger

Creating API Test Automation of a Service for Company X

Minh Ly

2018 Laurea



Laurea University of Applied Sciences

Creating API Test Automation of a Service for Company X

Degree Programme in
Business Information Technology
Bachelor's Thesis
May, 2018

Ly Trieu Minh

Creating API Test Automation of a Service for Company X

| Year | 2018 | Pages | 50 |
|------|------|-------|----|
|------|------|-------|----|

In project management, Agile is an incremental model, which is based on an iterative research and development method. As a result, multiple projects can be run simultaneously and they are divided into several segments to be used and changed according to the customers' planned requests. Of particular concern was that there was not any API test automation (TA) for shared services in company X.

The purpose of this thesis was to apply TA for a shared service called Area of the company and support their software TA workflow. In each deployment environment, this TA has been run to ensure that API endpoints have been called, responded and executed with results.

The objectives were to define a proper process for establishing, building and validating a TA environment. Thanks to the technical features included in enterprise editions of the applications such as JIRA, Bitbucket, Bamboo, Octopus and Confluence, all objectives were planned and finished successfully. API test automation was passed and test cases were documented on the system of company X. Therefore, the developers and Quality Assurance engineers have had opportunities to improve and develop features further. The knowledge base of this study was concentrated on API test automation and multiple functions of other technology components, which were used to structure a testing environment.

In this thesis, API test automation has followed the principals in X's Quality Assurance procedures. The test results have been verified by two different employees, including at least one software QA engineer. The Agile model was used as a research and development method, and JavaScript programming language, as well as Jasmine test framework were used. The test automation can be extended, tailored and developed further based on the company's needs.

Keywords: API Test Automation, Agile research and development method, JavaScript, Jasmine Test Framework, Testing Environment

Table of Contents

| | |
|--|----|
| 1. Introduction | 7 |
| 2. Scope and objectives | 8 |
| 2.1 Define process and requirement specification | 8 |
| 2.2 Build testing environment..... | 8 |
| 2.3 Validate testing environment | 8 |
| 3. Methodology | 9 |
| 3.1 Research and development methodology | 9 |
| 3.1.1 Agile..... | 9 |
| 3.1.2 Test automation in Agile | 10 |
| 3.2 Measurement methodology | 11 |
| 3.2.1 Time measurement..... | 11 |
| 4. API test automation | 12 |
| 4.1 API testing | 13 |
| 4.1.1 Unit testing and functional testing..... | 13 |
| 4.1.2 Web user-interface testing..... | 14 |
| 4.1.3 Security testing..... | 14 |
| 4.1.4 Load testing..... | 15 |
| 4.1.5 Runtime error testing | 15 |
| 4.2 API test frameworks..... | 16 |
| 4.3 API test tools | 16 |
| 4.4 Advantages and disadvantage of test automation | 17 |
| 5. Requirement specification | 20 |
| 5.1 Test automation goals | 20 |
| 5.2 REST API test automation rules | 20 |
| 5.3 Deployment checklist | 23 |
| 5.4 Ethical guideline | 23 |
| 6. Testing environment | 25 |
| 6.1 Create backlog | 26 |
| 6.2 Plan, design and develop test cases..... | 26 |
| 6.3. Review test cases on system..... | 27 |
| 6.4 Continuously integrate test cases | 27 |
| 6.5 Deploy test cases | 28 |
| 6.6 Document test cases | 28 |
| 7. Implementations | 29 |
| 7.1 Sprint 1: Agile test automation preparation | 29 |
| 7.1.1 Plan | 29 |
| 7.1.2 Design..... | 34 |

| | |
|---|----|
| 7.2 Sprint 2: Agile test automation implementation..... | 36 |
| 7.2.1 Develop..... | 37 |
| 7.2.2 Test and measure..... | 37 |
| 7.2.3 Release | 40 |
| 7.2.4 Feedback | 41 |
| 7.2.5 Maintain and document..... | 41 |
| 8. Evaluation..... | 42 |
| 9. Conclusion | 43 |
| Reference | 44 |
| Figures | 49 |
| Captions | 50 |

Terms and abbreviations

| | |
|------|---------------------------------|
| API | Application Program Interface |
| BDD | Behaviour-Driven Development |
| CPU | Central Processing Unit |
| DOM | Document Object Model |
| FTP | File Transfer Protocol |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| IoT | Internet of Things |
| JDK | Java SE Development Kit |
| JSON | JavaScript Object Notation |
| OAS | OpenAPI Specification |
| OP | Operating System |
| QA | Quality Assurance |
| REST | Representational State Transfer |
| RPC | Remote Procedure Calls |
| SMTP | Simple Mail Transfer Protocol |
| SOAP | Simple Object Access Protocol |
| SSO | Single Sign On |
| TA | Test Automation |
| TE | Testing Environment |
| TSO | Transmission System Operator |
| UI | User Interface |
| URL | Uniform Resource Locator |
| XML | eXtensible Markup Language |

1. Introduction

Company X is one of the leading power exchanges, in which power producers and buyers, regardless of their size or trading areas can trade, clear, settle and associate with their services across fifteen European countries. The company markets are operated from offices in Oslo, Stockholm, Helsinki, Copenhagen, Tallinn, and London. Since 2016, the company has thoroughly improved to deliver their technical capabilities, which centered on customer orientation. The company has integrated and enhanced their Application Program Interface (API) usability to accommodate more efficient and secured trades. Area service is one of the internal services that X's operators manage daily to operate many different trading databases such as trading areas, area types, markets, connections between areas and countries. Area service is perceived as the fundamental part of the Master Data Management System among X's internal services. With ongoing competition, cost pressure and market harmonization, the importance of maintaining a reliable Area service, which means assuring its stable APIs' performance, will increase the future members' satisfaction and enhance the key factor to act on new business opportunities. Company X strives to continuously strengthen its business by working with integrity, together with members and stakeholders, to achieve excellence. Hopefully this research will contribute to a consolidated Quality Assurance (QA) system to increase X's efficiency and growth opportunities.

This project was conceived during my time working for company X. As a QA trainee, it was found that there was a need to maintain the most practical and efficient QA procedure by automating the API tests of all shared services. The APIs of Area shared service are tested to make sure all internal calls of shared services and external calls of clients' services will reach the Area service successfully and rapidly. The thesis is an important property of company X that initializes economical Test Automation (TA) for entire shared services. A unified and centralized TA helps QA engineers among teams understand the services' performance faster. Therefore, all parties can achieve their business agility by delivering precise service quality and receiving speed client feedback.

2. Scope and objectives

This study focuses on API test automation of a shared service named Area, one of the core part of Master Data Management System. All passed test results, which should be deployed in the production environment, depends on the timeline of TA strategy of X. The test files should be documented and maintained on X's TA system for further use or investigation. Therefore, it is significant that API test automation of this thesis must carefully pass all X's Acceptance Criteria.

The main objectives of thesis are to support the consolidated QA system of X. It is intended to define a proper process and requirement specification to establish, build and validate a testing environment (TE). As a result, API test automation of this thesis is produced, tested and verified in that environment. This section has included these objectives, which are described as below.

2.1 Define process and requirement specification

The key aspect of delivering a reliable and stable TA is to define a proper process for its testing environment. Recently, there have been many different technologies, which have been used under enterprise editions to implement and deploy test cases in X. The technologies, which work based on some requirement specification and rules, are the constructive components to establish the process of TE.

2.2 Build testing environment

After visualizing the steps of testing process above, it is a dominant practice to start building the TE. This study should build the TE, in which developers and testers can easily spot the errors. The environment is also required to response the outcomes of test cases intuitively and accurately. By applying Agile methodology, it is critical that the study must be managed to build this environment in short preparation time, quick run checks and mitigation efforts.

2.3 Validate testing environment

After building the API testing environment, the last empowered practice is to validate that TE. API test cases are run to execute results in a demo environment, where other QA engineers can verify and give evaluation for this study. By doing this, API test automation of thesis can boost a consistent quality in alignment with the company's TA policy.

3. Methodology

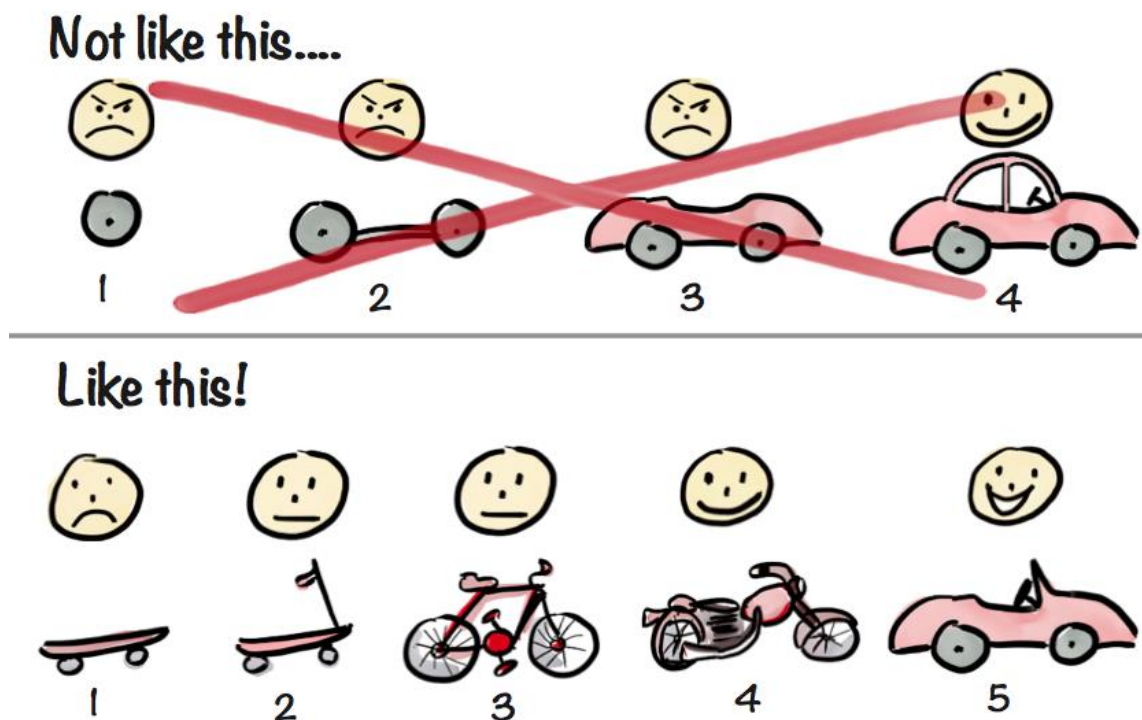
This section will describe the methodology of the research, development and measurement. As research and development methodology, the Agile method and TA in Agile are covered. Furthermore, time measurement methodology will be applied to analyze the outcomes of this thesis.

3.1 Research and development methodology

Research and development are the academic activities, which are chosen and designed to make the objectives of this thesis possible. The methods of these activities can be a set of processes that can be found in Agile model. Likewise, agile testing or TA in agile software development also has a full range of all functionalities that are formed by Agile method.

3.1.1 Agile

Agile is a practice including a set of values and principles that gives a continuous iteration rather than a predefined blueprint. The picture below can promptly show how Agile is illustrated in the research and development process.



Henrik Kniberg

Figure 1: How Agile methodology is illustrated (Hickerson 2016)

The first value of Agile is the individuals' interactions rather than processes and tools. For example, a team's daily meeting gives team members a foundation to make decisions how to develop their software incrementally. The second value of Agile is that the teams focus on working software rather than on comprehensive documentation. This is a professional measurement of conveying information within teams. The third value of Agile is customer collaboration rather than contract negotiation. In agile software development, the team must communicate continuously with their customers to receive feedback for improving better products' values. This routine activity not only harnesses changes in customer expectations but delivers the competitive advantage and final satisfaction to the software team. The last value of Agile is that changing requirements are welcome and fall into plans that bring the team many right metrics to achieve operational excellence. (Guru99 n.d.)

3.1.2 Test automation in Agile

The Agile method proposes an iterative and incremental approach to software testing life cycles. In Agile, business people and developers must work daily together throughout their projects. A TA project, following Agile methodology, is developed aligning with the software development. Every sprint is considered as one software life cycle, where the customers' feedback is listed and decisions are made. One sprint can take from two to six weeks, depending on different business strategies (Garg 2017).

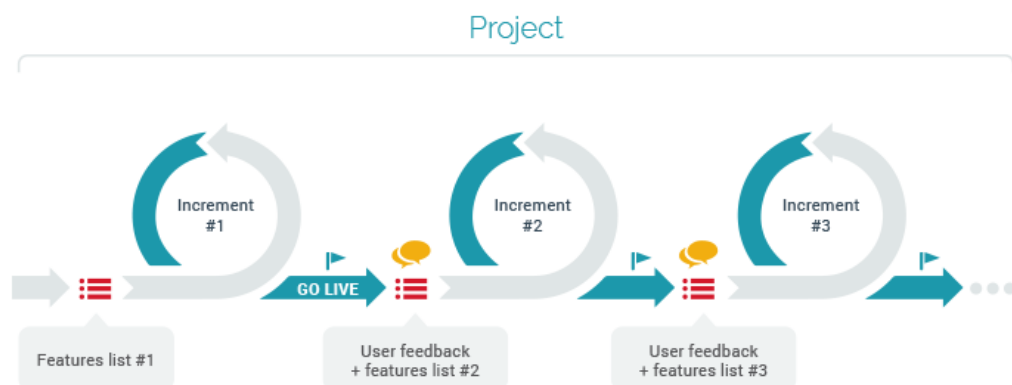


Figure 2: Project management in Agile (Garg 2017)

In each software testing life cycle, there are different segments such as Plan, Design, Develop, Test, Release and Feedback. First, in the Plan segment testers need to know about testing requirements, test tool selection, framework design and its features, resource or technology discovery, deliverables, milestones. Second, in the Design segment testers design the in-scope and out-of-scope items, schedule, timeline, levels or types based on the feature complexity, TA infrastructure. Third, in the Develop segment testers execute the test cases. The

prioritized requirements should be tested to return feedback to developers. Broken test builds are investigated by testers and sometimes by developers in need. Fourth, in the Test segment testers continuously test and improve automated cases due to required changes and updates. Fifth, in the Release segment after obtaining the expected TA quality, the test files should be deployed into product environment as planned. Sixth, in the Feedback segment the ongoing TA maintenance occurs. There will always be required a mixture of testing types and levels relative to the value provided by the available tests (Guru99 n.d.). Henceforth, more TA plans will be considered with much effective consultancy.

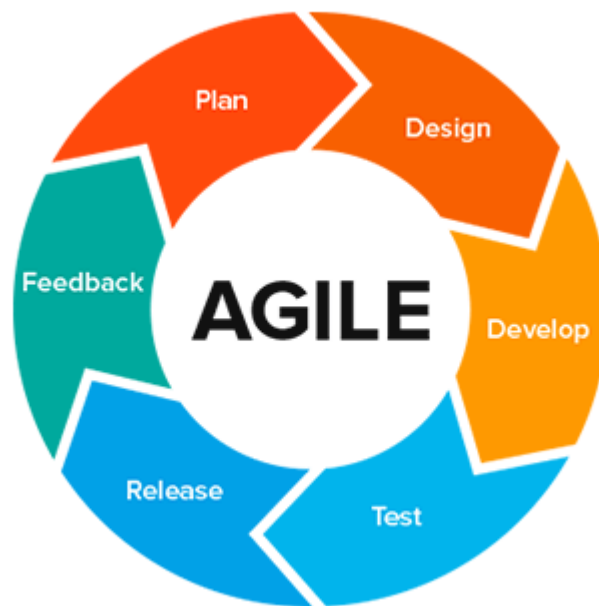


Figure 3: Agile software testing life cycle (OnlineBooksReview 2018)

3.2 Measurement methodology

As mentioned in the methodology section, this part will determine and analyze a measurement methodology called time measurement method. It is used to compare the difference between the time estimation of manual tests and the time estimation of automated tests.

3.2.1 Time measurement

Time measurement method brings calculated time-evidence from a digital clock that works as a time calculator. The clock has the digital format of Microsoft desktop clock, from desktop UI setting of Microsoft operating system (OP). The clock will use the unit, which is second, and calculate the amount of time, taken from manual tests and automated tests. This time measurement method is applied to evaluate the result of API test automation and support the

objectives of thesis. It means as a part of software development, agile TA helps testers maximize the flexibility and tailoring of their time. Details are given in regard to the method and results, obtained in the forthcoming sections.

4. API test automation

In this section API test automation knowledge bases such as API, XML, JSON, REST, SOAP, types of API testing, API test tools and frameworks are described. The knowledge base finds some TA factors in the implementation of this thesis. Besides, the advantages and disadvantages of TA are also mentioned to justify the current TA industry and prepare a foundation for the discussion part of this thesis that raises the challenges of TA all over the world.

API can be considered as business capabilities, or business logic, exposed over the internet for different applications to use. It allows applications to communicate with back-end systems. Normally these applications will return through their Hypertext Markup Language (HTML) web pages. When users react to a web page, which means they send a request on UI, that request is sent to the requested service database. Then, the service returns the requested data in either eXtension Markup Language (XML) format or JavaScript Object Notation (JSON) format. The reason is that database is returned not in a random formula but in their own structures. XML and JSON formats construct data and send data from the users or clients to the servers and vice versa. XML was developed in 1997. It uses identifying tags, like HTML and provides a rigid, hierarchical way of structuring data. JSON was developed in 2001. Because JSON is derived from the JavaScript language, it is easily human-readable. It can be condensed with fewer characters to be very lightweight. (SoapUI n.d.)

```
XML Example

<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

Figure 4: XML format example (W3schools n.d.)

JSON Example

```
{
  "employees": [
    { "firstName": "John", "lastName": "Doe" },
    { "firstName": "Anna", "lastName": "Smith" },
    { "firstName": "Peter", "lastName": "Jones" }
  ]
}
```

Figure 5: JSON format example (W3schools n.d.)

These two formats both help to pack and transport databases, the states of the objects. These resources are fetched based on the so-called CRUD, which orderly means Create, Read, Update, Delete. Over only the Hypertext Transfer Protocol (HTTP) methods, users can POST, GET, PUT, DELETE the resources. That is how the Representational State Transfer (REST) API works. Another type of API is Simple Object Access Protocol (SOAP) that invokes the services by the Remote Procedure Calls (RPC) method. On the contrary, REST API simply calls services via Uniform Resource Locator (URL) path. Not unlike REST API transfers database over HTTP, SOAP also does that work by using HTTP, Simple Mail Transfer Protocol (SMTP) and File Transfer Protocol (FTP). Nowadays, many companies moved from SOAP API to REST API. The reason lies in the fact that SOAP API performance is not as capable as REST API performance is. The advantages of REST API, compared to SOAP API, are the leaner code, easier calls from JavaScript and less Central Processing Unit (CPU) intensive. (Guru99 n.d.)

4.1 API testing

API testing involves testing application programming interfaces directly. As part of integration test, API testing determines if API logic matches the expectations of security, usability, reliability, testability and scalability. Since APIs lack a GUI, it is tested at the message layer. API testing becomes critical in TA, especially in the TA following Agile methodology because APIs themselves serve as the primary interface to the service or application logic (Guru99 n.d.).

Up to now, there are many types of environments and devices, which use API-based solutions. They are mobile applications, web applications, cloud applications, TV applications and Internet of Things (IoT) devices. They connect to, integrated with and extend software systems in a complex API architecture. Depending on the companies' strategy for testing their APIs there are different types of API testing to test the multiple tasks of their APIs. The types are API unit test, API functional test, API security test, API load test and API runtime error test.

4.1.1 Unit testing and functional testing

API testing types are usually known as unit testing, functional testing, web UI testing, security testing, load testing, runtime error detection (Guru99 n.d.). Unit testing is created when the

developers work on coding; they also code some unit tests. Unit tests are often explicit and predictable code-base. Because integrating unit tests is easy, unit tests help to find early problems that first do not seem like breakable, but eventually, they conflict with something else and result in a problem. The drawbacks of unit tests are that they consume quite much time and they cannot be run in the actual deployment environment. With functional testing, testers can test the functional demands, which are required in the necessary usability of services or applications. Because of that, one big advantage of functional testing is functional API test automation preferably conducts the cases in the similarly expected conditions. The disadvantage of functional testing will happen if testers make logical mistakes about the software usability or testers create some redundant test cases. (Guru99 n.d.)

4.1.2 Web User Interface testing

Web User Interface (UI) testing tests the application from a user's point of view. For example, when opening a web page, users input their usernames and passwords on a login page and click the sign in button. Another way of opening a web page is that users directly type the URL from the browser to open the web address they wish. Then, the web page shows up. In web UI test automation, testers often use a tool named Selenium. It is an open source, TA tool for web-based application. Selenium supports all the web browsers such as Google Chrome, Mozilla Firefox, Safari, Opera, Microsoft Edge and Internet Explorer. Selenium also supports all operating systems such as Windows, Linux, Solaris, Macintosh. It helps to test the web interaction when users click, input, select, navigate. With the help of scripts, Selenium also checks the expected response code. Web UI testing has its advantages when TA is done as the integration test, testing the entire end-to-end system in the way it is used. The disadvantage of web UI testing is that it tends to be the slowest and most brittle to execute. It takes a lot of time to spin up the browsers, talk to the back-end database and wait for the page to load. Web UI testing is fragile because as soon as developers change the UI, testers need to change the tests. Generally, Web UI test automation is done not for testing everything but for some specific scenarios. (BlazeMeter 2017)

4.1.3 Security testing

Security testing often includes penetration testing, fuzzing testing, authentication validation, access control and encryption testing (SoapUI n.d.). In security testing, API test automation ensures there is no security error in the APIs. Otherwise, the applications, which depend on the broken APIs, will entirely be in trouble. For example, a data breach in its back-end system can be a risky proposition, which requires quite in-depth security testing. It is a prevalent topic in organizations that the developers and testers must increase many security analysis skills and combine with their affordable security test methods to solve this kind of a prob-

lem. API security testing offers a lot of advantages in running threat analysis. The high-level automated API pen testing and other types, which include internal and external scales, demand various business costs based on a set of service variables.

4.1.4 Load testing

Load testing is a non-functional testing type. It is a testing technique to test the applications' behaviors not just under a massive amount of traffic, but how they perform every day with a real-time number of the user loads for the target of the application. With load testing, testers mainly try to understand the maximum operating capacity and bottleneck by increasing or decreasing the user quantity concurrently and incrementally. Load tests can be conducted in the usual or the peak time. It determines the load balancing of the systems adequately, and it evaluates which component of the systems cannot emulate the limited bandwidth accurately. Load testing should be done in TA unless testers will heavily measure and analyze how the bottleneck results with their manual methods. Sometimes specific scripts are used to implement, execute and verify the system. In most of the cases, when load testing is done manually or without appropriate tools, it usually adds up to the company's finance much more to increase their infrastructure. (SmartBear n.d.)

4.1.5 Runtime error testing

Runtime error testing is based on the concept of some TA types such as runtime error detection, exception testing, race condition testing, resource leak testing. A runtime error happens when the application is running, then suddenly its program crashes, terminates the expected execution. One of the possible reason is when users are supposed to input an integer data value instead of a letter data value. For example, after the wrong type of data value is input, the execution returns the red text, called stack trace, meaning that users have had an input mismatch exception. With the API conditions, runtime error TA detects to monitor how those mismatch exceptions will be handled. Furthermore, at the same time an application is executing, any thread interruption can take a chance, and those chances are called exceptions. By the time programmers develop those expected logic responses, sometimes they cannot control these exceptions. In exception testing, developers can initially set up rules with instructions to support the testers properly test the logic execution at the runtime. (Nielsen 2001)

Race condition happens when an application, having two or more operations in parallel, attempts to access a shared resource or to perform same thread execution. In race condition testing, testers use test frameworks to design the test classes in the ways of their test cases running against each other (Northcutt n.d.). For example, by default test cases in the same

test collection run one after another. With xUnit.net framework version 2, testers can write the test suites for each unit test (xUnit n.d.). By placing them in separate test classes, which means they are in different test collection, the cases can be run simultaneously. xUnit framework works as a collective unit test framework, which is highly structured and object-oriented. As another example, race condition testing can also happen when testers run the test methods at the same time and the test classes of those methods in sequence. With TestNG, a Java testing framework, testers can simplify the annotated test methods and use the testing suite, an XML file, to write and execute a number of test cases in parallel (Reddy 2015). These two examples allow testers to run the race condition TA.

4.2 API test frameworks

Test frameworks are used to structure the test case coding at some standardized TA designs. Test frameworks are robust methods, which support TA technique with organized implementation, risk management and operation maintenance rather than randomly customized and saved codes. When choosing test frameworks, testers also need to question many critical measures. It is recommended that the framework should be made for testers' customization so that they can limit or expose their testing intentions permanently as well as cope with the team's long-term visions. Like test tool selection, it is a common sense for testers to make sure both test tools and test frameworks will be supported. That is to say, the tester will not be the only person doing this testing because only the tester knows how to use those tools and frameworks in the team. (CrossBrowserTesting 2017)

In the TA framework market, there are more and more simplified, constructive and open-source testing frameworks with their advanced guidelines. For instance, some well-known test frameworks are Selenium WebDriver, Protractor, Jest, Tape, Robot Framework, Capybara, Jasmine, Mocha, Watir, Appium, Cucumber, Lettuce. All test frameworks require testers prior knowledge of coding, processing and browsing based on the diverse usage. Test frameworks assist testers with their architecture in particular. It can either be keyword-driven, behavior-driven testing frameworks, hybrid testing framework, module-based testing framework, library architecture testing framework and data-driven testing framework (Power-Morse 2016).

4.3 API test tools

API test tools are used to test API requests based on their set-up methods. The tools are competent to return response headers, response bodies, and response codes. The tools are also supposed to provide testers a human-readable working frame to save the expected outcomes. Some API test tools even supply professional collections, environments, categories to produce

and to be saved in good fractions for the next or consecutive test cycles. Then, testers can shorten their time when they manually input different values in every tested request. In the API test automation market, there are more than twenty popular API test tools, which promote all types of TA strategies, goals, budgets, future proof. Some well-known API test tools are Postman, Karate DSL, REST-Assured, SoapUI, Hippie-Swagger, WebInject, Pyresttest and Runscope.

There is no best API test tool, but there are some most appropriate API test tools, which serve the needs of the testers and their teams. When selecting test tools, testers need to consider many different activities. For the most of reasons, the tool should be tailored for the tester and the team's condition, thereby integrating tightly with the current developers and testers' ecosystem. The test tool should be chosen from the proof of future and it should let the test frameworks be involved. Therefore, when the potential issues of the application grow with time, the selected tools still work in the product roadmaps. For example, when testers choose Selenium, which is great for web-based automation, then the team comes up with testing the service APIs. In this situation, Selenium will not be able to fulfill the test tasks. It maybe REST-Assured will support for those. There are also some non-open-source vendor-based test tools that the teams can have more functionality options such as Tricentis' tool or SmartBear's tool. They cover a lot of TA types such as a mainframe, API, web services.

4.4 Advantages and disadvantages of test automation

In the past twenty years, TA started its viral symptom due to its incredible impact on the test community. In fact, although some testing software providers have globally overstated their tools' functions for the commerce, almost all of the testers have locally heard and experienced some good and bad sides of this automating technology. In this section these two aspects will be analyzed in the scope of how test automation effects on human and machinery.

First, regarding the TA benefits, testers all know TA helps them in saving time and technical management. After essential set-up and a smooth test cycle run, testers can quickly improve their self-testability or turn to gain more skills on other tasks. This is only true when the testers understand how-to prepare in the planning phase, not to say in their tool and framework selection. As discussed in the tools and frameworks sections above, testers must consider the current situation of their teams. For example, when choosing the TA tools and frameworks, if testers make sure the tools and frameworks work well with the software system, programming languages that their teams are running, testers can quickly manage their TA projects. (Guru99 n.d.)

Second, compared to manual test, automated test ensures developers and testers quickly share understanding and ownership because TA requires the testers' coding or scripting skills. TA gives testers higher-level of technical promotion. Moreover, with the integrity and competence, testers communicate with programmers to create an agile culture. Thus, the observability and controllability of the TA will become more flawless with less endeavor.

Third, the transparency of TA helps testers work independently on the testing projects. When anyone is interested in the testing results, they can sign in to check the TA system. This practice rarely happens in manual testing. The visible results with the systematic error-messages are valuable and convincing for developers and project managers to take actions. At last, TA plays an important role in many software organizations because of its cost-effectiveness. Automation tools are costly for business, but if the software strategy leans a long-term investment forwards, it is worthy that the corporate performance should try to apply TA. The enterprises' resources need to assure the balance between TA capabilities and its benefits.

In contrast to TA advantages, its drawbacks are that the explosion of the IoT and cloud technology have made this automating industry more costly, more complicated, in the parallel of software development tools and methodology growth. Therefore, TA has three potential human-related threats that possibly cause the wrecks in many cases. The first disadvantage is indefinitely chasing the crowd or chasing the hype situation. Some test-software companies have invented newer or better TA tools, TA frameworks with the hope to prevail business competition. Other companies, who buy to use the tools and frameworks, also have their organizational impacts on planning, designing, developing, testing, measuring, implementing, maintaining. The entire ones, who insist on the agile-type methods and have the increased changes in the software features, find their TA constraints less painful in that mushroom after rain symptom. (Horne 2014)

The second disadvantage is that the TA quantity goes faster than its quality. After many years testers all over the world have experimented or implemented the new and improved TA products. It turns out that some rafts of commercial TA tools are undeniable. Some low-cost and open-source TA companies sensibly offer the basic products with quite trivial functions or only pass-or-fail-type approach. Without the regular overhead purchases, the tools are unable to be fully-functioned or upgraded. This hot monetization strategy happens between the TA software products and the software service companies. Only those highly professional companies, who have long-term software investment, dedicated to one or two TA enterprise resources find themselves a stable stand in the automating technology storm. Additionally, TA tools present users with a bunch of new compatible versions to achieve the pervasiveness of their tool technology. For example, Selenium is a handy web-based TA tool that likely confuses all TA starters from now and then. Since 2004, Selenium has had seven different named

versions for some specific upgraded purpose. They are Selenium IDE, Selenium 1, as known as Selenium Remote Control, Selenium 2, as known as Selenium WebDriver, Selenium 3, WebDriver, Selenium Grid, Selenium. Unfortunately, after more than twelve years, because of its lack of supported operating systems, browsers and platforms to run extensive UI tests of some thick-layered applications, a new docker-selenium node approach has launched with its name, Zalenium. As the producer's commitment, Zalenium increases all capabilities that Selenium Grid has not figured out at the moment. It is the latest unofficial Selenium project, aiming to provide a simple Web UI test automation grid (Zalando n.d.).

The third disadvantage is that all TA tools and frameworks require testers from medium to long absorbability-time with solid technical skills. Every TA deploying change can be unviable or degrade the business agility because of the long set-up time into the whole enterprise system. The software companies need to be careful and fast in training their QA teams. If the testers are using the enterprise-wide TA applications, whose vendors provide firm support and guidance whereas the freebies will receive those values from groups or forums. On the other hand, TA has been unable to replace the manual testing yet, because TA does not operate entirely the extremely complex test cases, which only human can brainstorm, as often seen in black box testing. (Eriksson 2013)

5. Requirement specification

The following parts move on to describe in greater details the requirement specification, which includes TA goals, REST API test automation rules, deployment checklist and ethical guideline. These sections are written based on the acceptance criteria of X's Quality Assurance competence. The API test automation of thesis must follow this corporate knowledge to have its outcomes be objectively assessed.

5.1 Test automation goals

There are five suggested goals that the API test automation must reach:

- The first goal is that API test automation has to increase development speed and deliver fast quality feedback.
- The second goal is that all test cases must provide confidence that every build passes the TA environment must be deployed to the production environment.
- The third goal is that API test automation ensures the reliability and stability regardless of features' changes.
- The fourth goal is that all automated tests cover the acceptance criteria.
- Finally, the fifth goal is that API test automation project must be designed and documented in a shared repository.

5.2 REST API test automation rules

Before proceeding to the API test automation rules, it is important to explain some technical terms such as 'spec', 'endpoint', 'matcher', 'custom matcher'. In TA terms, 'spec' is defined as a technical specification, which explicates a set of requirements for a given code. 'Endpoint' is a unique address of the World Wide Web page on the Internet. It takes users to data resources and users interact with those resources by different methods such as get, put, post and delete. 'Matcher' is a technical standard, which is "used to search through a text for multiple occurrences of a regular expression" (Jenkov 2017). Accordingly, a 'custom matcher' is used as intentionally customized matcher that helps to remove code duplication in specs (Cobb 2014). Turning now to REST API test automation rules, there are five rules, which are described as follow.

First, one spec should derive from one endpoint and one method (GET, PUT, POST, DELETE). The reason is that it is easier to link specs and REST API reference and review the TA coverage by looking at specs list. The spec names should look like: endpoint URL + method. Sometimes the same endpoints could be external and internal without authentication. It is unnecessary to test the same things twice. In that case, the public endpoints will be covered with

all test cases and added one or few positive tests to the same spec for internal endpoints only to make sure it works without authorization.

Second, all responses must be verified from requests with custom matchers. To keep the APIs consistent, tests are expected the same response from different endpoints. It is better to give it with values of custom matchers so it would be the only source of the expected result. On the other hand, when the response is in json format, Jasmine test framework did not provide good visibility of errors. The validation of its json schema locates in a shared json schema definitions in the company's shared folder. Using custom matcher `toConform()` to verify json schema is also a correct way to do. (Jasmine n.d.)

```
//instead of
expect(result.status).toEqual(405);
expect(result.body.Message).toBe("The requested resource does not support
HTTP method 'PUT'.");

//do like
expect(result).toBeMethodNotAllowed(method);
```

Caption 1: Using custom matcher in API test automation

Third, tests must be verified data in its response, not in its structure. It is often when the response contains a correct data structure, format or the number of parameters, but data itself is wrong. If the data is changed for some reason, QA engineers must try to get rid of it and create another isolated account to generate test data in spec or to have predefined data in the database.

```
function distinctBy(parameter, json) {
  return _.uniq(_.map(json, parameter));
}
...
expect(distinctBy("Ccp", result.body)).toEqual(["N"]);
expect(distinctBy("TradeStatus", result.body)).toEqual(["Completed"]);
...
```

Caption 2: Predefined data in database

Fourth, `for` loop must be used to iterate the same call with different input or output data. It is required to make the same call for different input or output data, for instance, device operations. It makes sense to create the array of input or output data and run the same `it()` for each element. The array of input or output data gives much more visibility to review test coverage, but fail report must point to the correct array element. On the other hand, there

are disadvantages of that approach so that it must be ready for resistance in Production environment and proposal advocacy. For example, it does not make sense to create with less than ten test cases such as:

```
notAllowedMethods.forEach(method => {
  it(
    "WHEN called with not allowed method " + method + " THEN status 405
and error 'The requested resource does not support http method " + method
+ "'.",
    done => {
      trades()
        .callWithMethod(method)
        .then(result => {
          expect(result).toBeMethodNotAllowed(method);
          done();
        })
        .catch(catchers.printAndFail(done));
    }
  );
});
```

Caption 3: Example of called method iteration in API test automation

Fifth, Builder pattern must be used for endpoint request function. The pattern below lets functions be extended easily in future and add query parameters, request methods like PUT, HEAD without exporting new functions.

```
const trades = token => {
  let qs = {};
  const options = common.getOptions(api, endpoint, qs);
  if (token) options.headers.authorization = "Bearer " + token;
  const tradesOptions = {
    fromDeliveryHour: function(utcDateTimeString) {
      if (typeof utcDateTimeString !== "undefined") op-
tions.qs.fromDeliveryHour = utcDateTimeString;
      return this;
    },
    toDeliveryHour: function(utcDateTimeString) {
      if (typeof utcDateTimeString !== "undefined") op-
tions.qs.toDeliveryHour = utcDateTimeString;
      return this;
    },
    forAreaCodes: function(areaCodes) {
```

```

        if (typeof areaCodes !== "undefined") options.qs.areaCodes = area-
Codes;
        return this;
    },
    get: () => call(options),
    callWithMethod: method => {
        options.method = method;
        return call(options);
    }
};
return tradesOptions;
};

```

Caption 4: Example of Builder pattern in API test automation

5.3 Deployment checklist

In this thesis, another requirement specification is TA deployment checklist. When companies establish product deployment checklist, it yields benefits in release commitment. Likewise, a TA project with clear deployment checklist is a wise arrangement that assesses almost actual problems. Furthermore, it reduces continuing demands for maintenance and failure analysis before and after a release. The API test automation of this thesis must orderly follow this deployment checklist instruction. For instance, before product deployment testers must verify that there is no message in error queues. If there are any messages, testers must requeue them and check whether or not they are processed. If there is no message, testers continue to abort the rest of deployment process.

5.4 Ethical guideline

As discussed above, this section will state and explain the ethical issues, which are approached in the API test automation of this thesis. It is critical and judicious to set up an ethical guideline as one of TA requirement specifications. The reason is that there is a significant correlation between software development and software testing. When programmers are guided on software creations to make the programs work, “testers focus on failure because it improves their chances of finding it. Look for key problems in product with all your creativity and skill” (Bach 2002, 6). An ethical guideline in software testing not only relates to technology aspect but to social and personal aspect as well (Stahl 2018). This section is written based on X’s ethical guidelines combined with the self-reflection on Software Engineering Code of Ethics (IEEE-CS/ACM n.d.).

As an employee in the X Group, a tester shares a common responsibility for maintaining X's general standards of integrity and accuracy. It helps to ensure that all work performed at X is conducted in a prudent manner. Integrity and professionalism are keywords in an organization such as X. This means that a tester must act with integrity, honesty and impartiality. It is necessary for a tester to handle information in professional secrecy (X's Ethical guidelines 2018). The content of this thesis is reviewed by at least two supervisors from X to conduct its integrity. The TA result of thesis is stored in the shared TA project of X and verified by the QA team members to conduct its confidentiality.

Regarding self-reflected on software engineering code of ethics, the TA of thesis acts consistently with the TA needs of X. In the best effort to bring benefits to the company and clients, the TA of thesis provides API test automation of a shared service named Area, which plays an important business logic testing role. A tester ensures that API test automation of thesis can present related modifications to meet the requirement specification standards. As mentioned above, the tester maintains integrity and independence in the judgment or evaluation of thesis. Moreover, the tester achieves the objectives of thesis in mutual trust, cooperation and openness in the relations with colleagues. The tester participates in long-term development and contribution regarding the practice of thesis and promotes an ethical approach in this profession. That is to say, the tester should be able to improve API test automation of thesis and contribute to further useful and qualified TA at reasonable cost and within reasonable time if being asked (StackExchange n.d.).

6. Testing environment

Following are steps that establish a TE for API test automation. They are defined as developing components, which process in each environment or platform for different purposes such as creating a backlog, planning, designing, reviewing, continuous integrating, deploying and documenting test cases. For each purpose, a tool or an application is used to analyze and track the processes. They are such popular and useful applications that almost software enterprises invest and apply these technologies in their daily development tasks. They are JIRA, Visual Studio Code, Bitbucket, Bamboo, Octopus and Confluence.

First of all, in JIRA application a backlog is created to store technical requirement specification and track process in the form of a ticket. Second, test cases are planned and designed carefully in Visual Studio Code, a programming application. The cases should follow REST API test rules, X's Acceptance Criteria and requirement specification. In this phase, Swagger and Postman tools should also be used to check the validation of the API endpoint manually. Third, all coding tests are pushed to a code review system called Bitbucket to be viewed and merged into the whole TA system. Fourth, test cases are deployed in a continuous-integrate environment called Bamboo. Fifth, test cases are deployed in the next automated deployment server, where multiple services are managed. This automated platform is called Octopus. Sixth, after being deployed successfully in Octopus, test cases are documented in a multimedia content application called Confluence. By way of illustration, the steps to establish TE is made in a workflow hereunder.

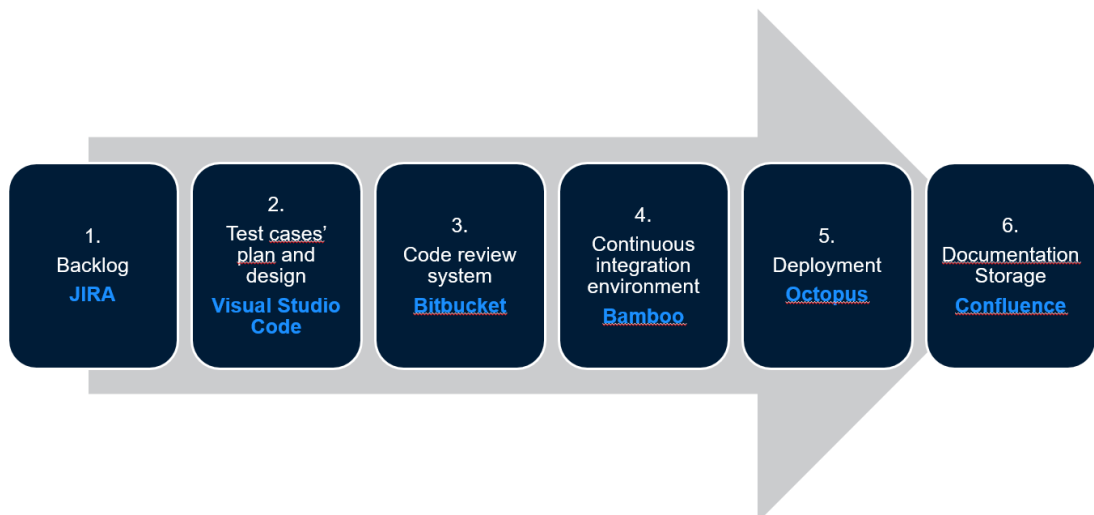


Figure 6: Testing environment workflow

6.1 Create backlog

JIRA is an application that helps software teams integrate their systematic bug-tracking and customized milestones within multiple projects. With the features such as Backlog, Scrum board, Bug board, status, priority, component, label, affect version, fix version, epic link and notification alert, JIRA is the best choice for tracking API test automation of this thesis. JIRA helps tester commit, control and resolve the test tasks in every sprint. In this context, JIRA is the platform where tester can start and end the assigned TA tasks with Agile methodology (QASymphony n.d.).

6.2 Plan, design and develop test cases

In this phase before planning and designing test cases, Postman and Swagger, the well-known API test tools, should be used to manually check if the database or information of API endpoint is valid and reachable. Then, test cases are planned, designed and developed straight into a programming application. It is Visual Studio Code, a text editor, which is sponsored by Microsoft.

Postman is a simple and convenient tool, which is used for API testing. It has an executable API description with collection format that allows users to run requests, test, debug, create TA, mock TA, and document API testing. In this thesis, Postman is used to manually call internal and external API requests. It also helps tester check the responses and verify whether or not the correct data is written to the database on PostgreSQL (Wagner 2014).

Swagger is the world's largest framework of API developer tools for the OpenAPI Specification(OAS). It helps in designing and documenting the entire API lifecycle. Programmers and testers can use the information, which is created in Swagger to test and deploy the APIs. (Vasudevan 2017) In this thesis, Swagger is used to establishing the internal and external API endpoint documentation. With the API description format, tester can test the API endpoints of Area service manually by clicking the button called "Try it out!" on Swagger. Since then, tester can understand how the database needs to be defined in TA suites. These databases should be defined, grouped in relevant test with proper spec names (Rosenstock 2017).

Visual Studio Code is an accessible and well-designed text editor application, which is created and developed by Microsoft since 2002. It contains hundreds of extensible and customizable features, extensions and settings, which smartly support and comfort the programming projects of developers around the world. It is a cross-platform, used in different operating systems such as Windows, Mac and Linux. In this study test cases are planned and designed in Visual Studio Code application, in which programming language is JavaScript and test frame-

work is Jasmine. This language and framework selection is agreed upon after tester had TA counseling with QA project leader. A centralized TA project helps to perform the business agility and cost-effectiveness.

JavaScript is an object-based and human-readable programming language that is used to make web pages interactive. It is known as an interpreted language. It does not have to be compiled. It is executed on the client's computer or in the browsers. With JavaScript, users do not need a server somewhere else but run the code locally on users' computers. They simply need to create a .js file and include it in the HTML codes.

Jasmine is a behavior-driven development (BDD) framework for the testing JavaScript code (Jasmine n.d.). With the BDD, testers directly link up the acceptance criteria from the requirements of software development stories to their test cases. Jasmine test framework neither depends on any JavaScript frameworks nor require a Document Object Model (DOM). It has a clean, obvious syntax such as `given`, `when`, `then` that provides users the ordered functions follow as `describe`, `it`, `expect`. Testers can write some `describe` to organize multiple test suites and test cases. Then, actual test steps are written in `it`. Each test step stays in each `it` function, which depends on different positive or negative test scenarios (Ben 2017).

6.3 Review test cases on system

Bitbucket is a self-managed Git solution tool, providing "source code collaboration for professional teams of any size, across any distance" (Bitbucket n.d.). Bitbucket has a superior code review system that all branches, discusses, merges, pull requests stay in a turnaround time control. In this thesis, Bitbucket is used to push and merge the created gits of API test automation of thesis. By using the enterprise version, Bitbucket Data Centre powerfully supports developers and testers many different Git projects' development at the speed of LAN and professional workflow control (Lesunova 2018).

6.4 Continuously integrate test cases

Bamboo is a continuous-integration environment that whenever new codes are pushed to the repository, the entire system reacts instantaneously with rapid identification. In this thesis, Bamboo is used to ensure pushed tests to be systematically integrated with the existing TA codes. Therefore, any errors occurred when integrating both new codes and current codes would be notified and fixed immediately (Bamboo Support n.d.).

6.5 Deploy test cases

“Octopus Deploy is an automated deployment server that makes it easy to automate the deployment of ASP.NET web applications, Java applications, database updates, NodeJS application, and custom scripts into development, test, and production environments” (Octopus n.d.). In this thesis, Octopus has a role as an automated deployment environment. After the test cases are pushed into Bamboo application, this Octopus environment-control-platform automatically deploys the test files into its first environment called Automation Test. Then, the test files are verified to be pass-checked by QA engineers and they will be manually deployed in the next environments which are Development, Quality Assurance, Pre-Production, Member Testing and Production.

6.6 Document test cases

Confluence is an interesting multimedia content tool that allows users to create, organize and manage many project documents or guidelines. It extremely fits in product managers or developers’ needs thanks to its embedding diagram features that brings intuitive page layouts. In this thesis, Confluence helps in researching information of all X’s documents such as ethical guidelines, TA procedures, TA rules, X Test Automation project and the enterprise architecture of X. (Keough 2017)

7. Implementations

The implementation of this thesis can be discussed in two phases, which are preparation phase and implementation phase. They happen in two continuous sprints. In the first sprint from April 16 to April 27, the Agile TA preparation of API testing is carried out. In the second sprint from April 30 to May 11, the actual API test automation implementation is done that includes the activities such as development, test, measurement, release, feedback and maintenance.

7.1 Sprint 1: Agile test automation preparation

In this section technical background, scope, objectives and requirement specification are looked into to make a TA plan and a TA design of this thesis. The precise and feasible TA plan will help to proceed the TA implementation efficiently. With Agile methodology, these two activities, planning and designing, are critical processes because they define what to test and how to go on it within a sprint cycle.

7.1.1 Plan

Test plan section can be viewed as documented intentions of a process for running TA. API test automation of this thesis is intended to have all passed results and the passed tests must be deployed into the production environment. It means API test files must be deployed and checked in each environment such as Test Automation, Development, Quality Assurance, Pre-Production and Production. Thanks to the functionalities of tools and deployment environments such as JIRA, Postman, Swagger, Bitbucket, Bamboo, Octopus and Confluence, API test automation of this thesis is processed in a professional systematic procedure. With JIRA, a TA request of API endpoints is created at the beginning of a sprint. The request is a ticket, which requires creating TA API Area GET /api/v1/deliveryareas endpoint.

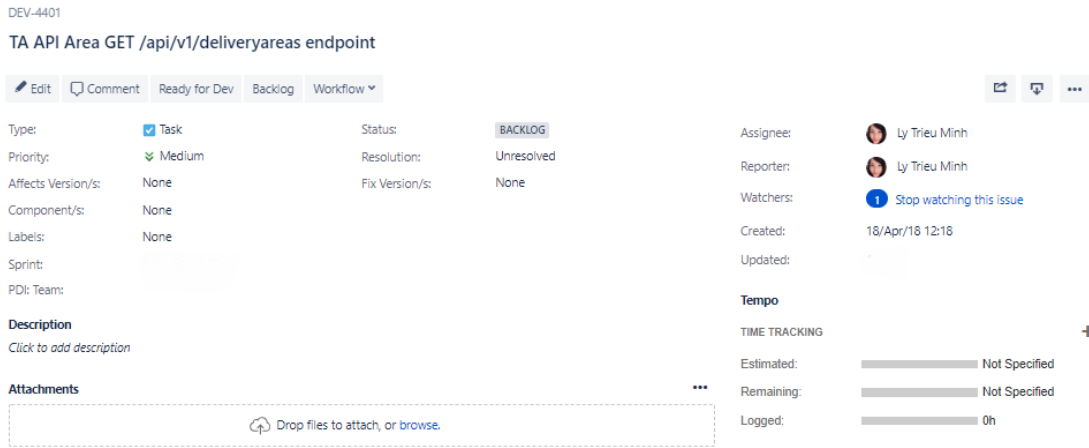


Figure 7: Test automation request on JIRA system (X's JIRA 2018)

JIRA has the purpose to collect the technical requirements to assign tickets to developers or testers. Some tickets have technical requirements, described in the description box. Some other tickets do not need a description because it depends on the team’s agreement. The ticket above requires a TA to test an API endpoint called `deliveryareas` of Area service. JIRA is used by the development team only. The company’s business units do not have access to JIRA. The technical system owner and the release manager can also access JIRA to see progress and status of this ticket. JIRA tracks how this ticket issue is resolved and gives the status of “To do”, “In progress”, “Implemented” or “Done” to this ticket. The JIRA workflow below explains that after being created, API test automation is started that makes this ticket have In progress status. Since API test automation is still in planning phase, this ticket has To do status. The entire statuses are orderly changed thereby the second sprint starts.

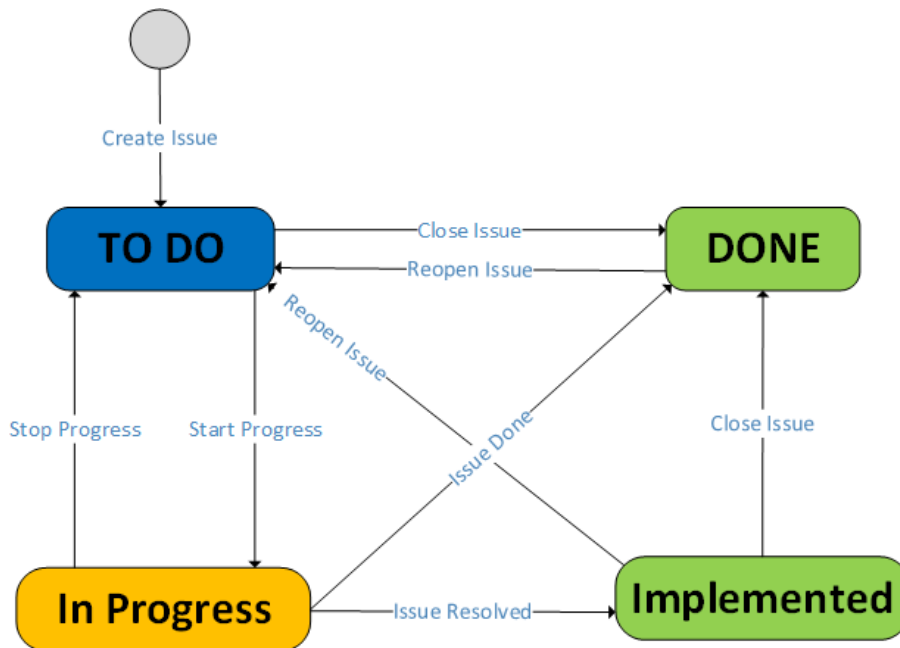


Figure 8: JIRA workflow (JIRA n.d.)

After creating a systematic request on JIRA, API test automation is planned to use Bitbucket as a developing environment to clone the Test Automation project from X's Bitbucket system to the developing program in a local laptop, which is used to code the test files. According to the requirement specification, API test automation of thesis needs to use the shared folder, which is X Test Automation for shared libraries and scripts. It means that test files should be created and stored in the rest-api folder of X Test Automation project since they are started.

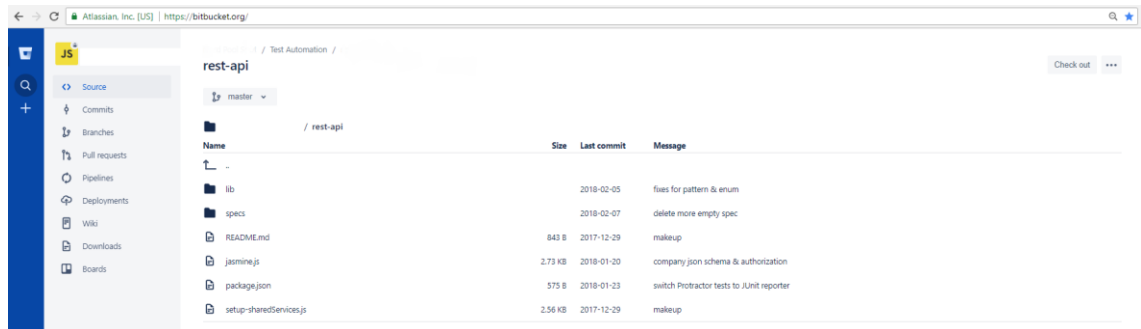


Figure 9: Test Automation project on Bitbucket (X's Bitbucket 2018)

After cloning the repository of the project to the local laptop, API endpoints, which is stored on X's Swagger site, is tested directly on Swagger and also manually checked by Postman tool to make sure all input database of Swagger is valid. Area service has its API endpoint documentation stored on Swagger editor program. The photo below shows the API endpoint of Area in Area service that is going to be tested.

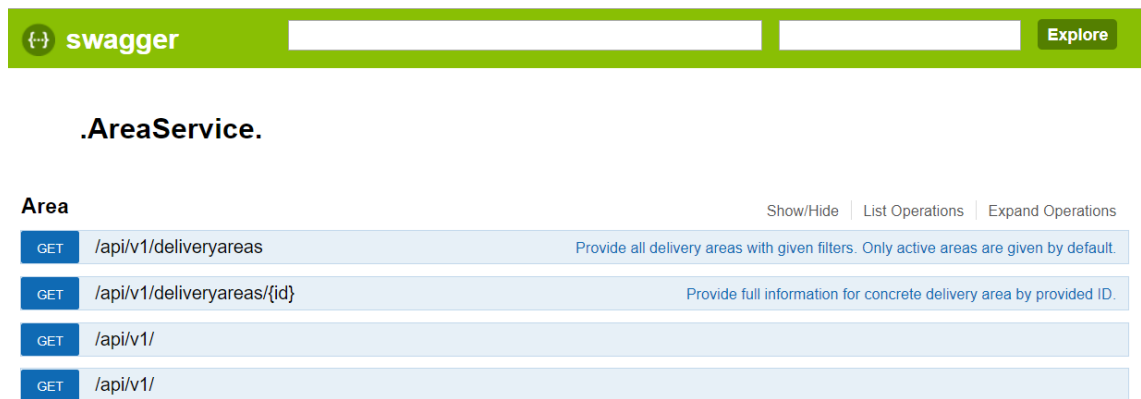


Figure 10: API endpoints of Area service to be tested on Swagger (X's Swagger 2018)

After testing the validity of database on Swagger, API endpoints can be checked by Postman tool to verify one more time if the endpoints can be called successfully.

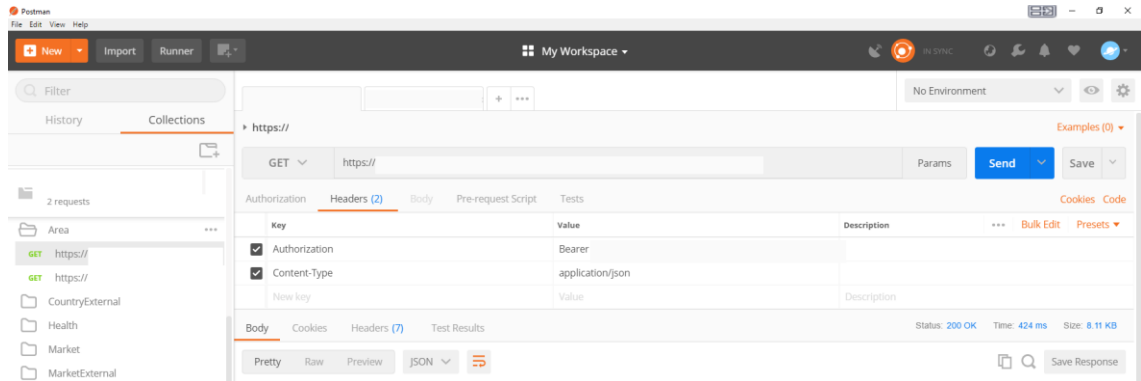


Figure 11: First API endpoint of Area service is checked by Postman tool (Postman n.d.)

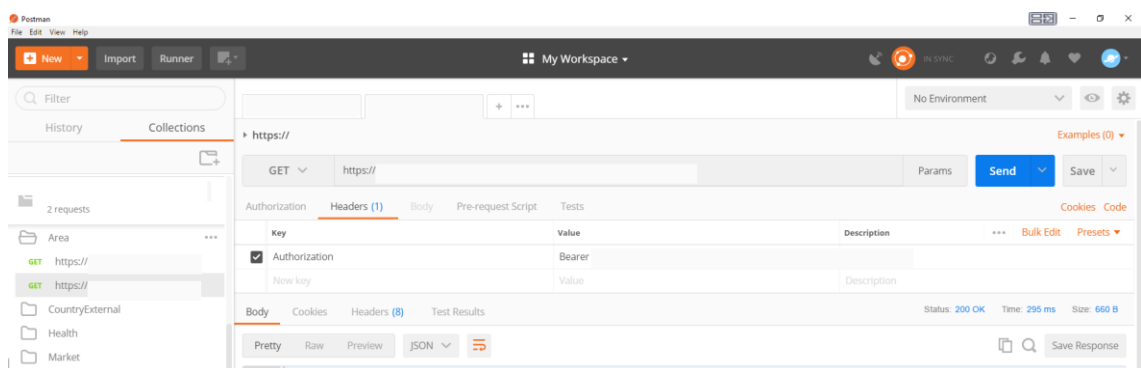


Figure 12: Second API endpoint of Area service is checked by Postman tool (Postman n.d.)

Before doing manual tests on Swagger and Postman, they require the authorization check by asking users to input a valid bearer token. A two-factor authentication service of X, called Single Sign On (SSO) service, is responsible for issuing this token.

SSO service authenticates users once to access multiple applications during two hours. In the first logging time to an application, SSO service receives the logged credentials of users from web UI. From the fetched database, users can obtain a valid token. It means during two hours since the token is created, users can access other applications without typing their usernames and passwords again. The purpose of this token-based authentication is to provide individuals fast and easy access to all restricted applications that users hold access permissions. SSO service supports users, who are operators, to manage and change the user rights in User Manager service. An overview of how a token can be requested and used is illustrated below. (Foppa & Pehkonen 2017)

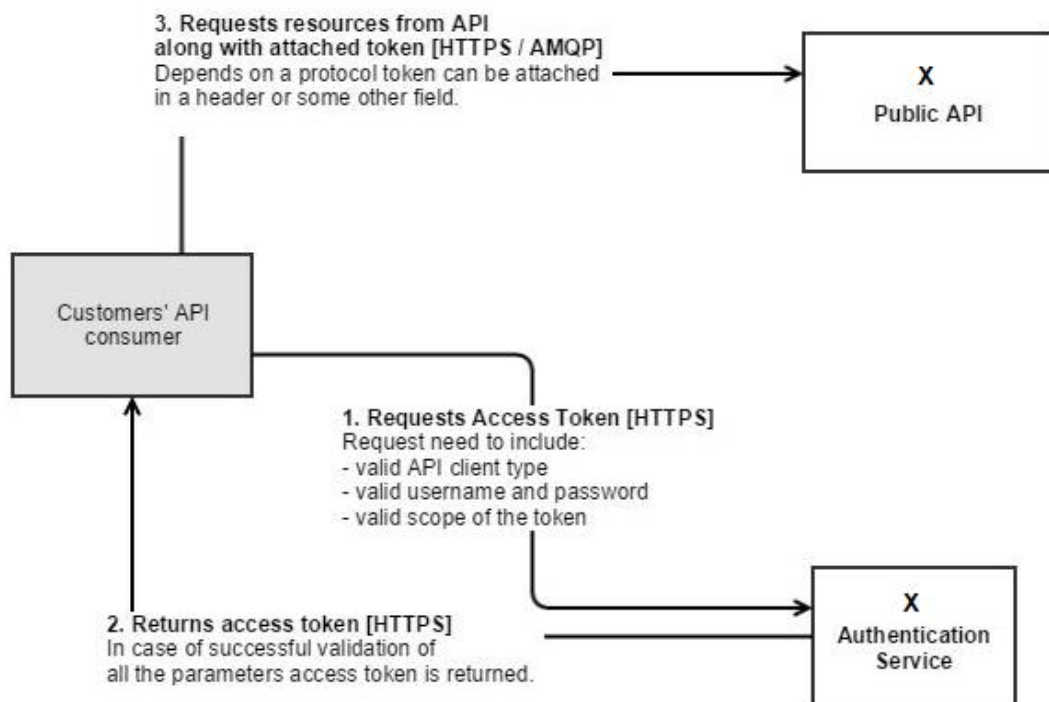


Figure 13: Token access flow (X's Single Sign On 2017)

After making sure the API endpoints have successful manual tests with Swagger and Postman, a TA environment is set up based on the instruction in README.md file of X Test Automation project. Visual Studio Code application is recommended to be installed. Then, TA environment needs to install NodeJS, all its required packages, Protractor and Java SE Development Kit (JDK). The reason for NodeJS and its package installation is that API test automation of this thesis uses JavaScript programming language to write test cases. Protractor and JDK are used to bind Selenium WebDriver for Node.js. Therefore, TA environment needs to update the web driver-manager to control Selenium WebDriver. This WebDriver's goal is "to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems" (SeleniumHQ 2018). Configuration steps are planned as below:

Step 1: Install Visual Studio Code application from <https://code.visualstudio.com/> It should be downloaded with Windows x64 version.

Step 2: Make sure NodeJS to be installed by going to the location of the `rest-api` folder and run `node --version` in the terminal

Step 3: Install all required packages by running `npm install`

Step 4: Install the global version of Protractor by running `npm install -g protractor`

Step 5: Make sure TA environment to have JDK installed by running `java -version`

Step 6: Update web driver-manager to control Selenium through it by running `webdriver-manager update`

Step 7: Start Selenium driver by running `webdriver-manager start`

After setting up the environment properly, specs are created based on the naming rules that looks like endpoint URL + method. Test (`its`) and test sets (`describes`) had to be isolated and were not dependent on each other. Once test cases in the library are ready to use in different TA environment suites, they have to be moved to `common` folder. The direct push to the master branch is allowed only for the quick fix of a failing test. When adding a new spec, test, lib, config or function, a branch and Pull Request to push changes had to be created. The test cases have to avoid becoming dumb or duplicated. The reason lies in the fact that running the same set of tests for different conditions will waste resources and bring up maintenance problems. In any case, pending tests should be linked to JIRA tickets.

Furthermore, tests, which make the TA environment fail for a long time, are not allowed to commit. The reason is that they cause lots of waste from pointless investigations by other people after they are deployed in further environments such as TA, Development and Quality Assurance. The passed tests should not produce much information but only indicate that they pass. The test failure should give further information in the console for testers or developers to find and verify. Once being deployed in further environments such as Quality Assurance, Pre-production, Member Testing and Production, the investigations of broken builds are usually treated by the whole development team and DevOps team. Last but not least, automated tests should never replace the necessary manual tests entirely.

7.1.2 Design

Having defined the strategy how test plan is approached, the moving on section discusses a test design for API test automation. It includes the details of conditions and ways that test suites are written. This activity helps the tester specify the requirements and mitigate errors in the scope of overarching goals. Designing test cases requires certain knowledge about system behavior. It may consist of functional design, user administrative procedures and system requirements.

Based on the requirements of energy trading, an area represents a control zone as part of an energy network. It is operated by a Transmission System Operator (TSO) for power trading or another entity, which is entrusted with transporting energy on a regional level in a fixed infrastructure that is natural gas or electrical power. Each delivery area is assigned to a market area, which represents a greater uncongested price area, like a national grid, or other high-

level control zones of a power network which can contain multiple delivery areas (X's XBID 2014).



Figure 14: One delivery area is assigned to one market area (X's Arter n.d.)

A delivery area embodies an area inside a market are. When entering orders, a delivery area must be specified to which a bought commodity is delivered, or from where a sold commodity is delivered. Then, each trade record will contain the outgoing and receiving delivery are (X's XBID 2014). In API test automation of this thesis, the expected results of test cases are determined and documented with JavaScript programming language in Visual Studio Code application.

Beside that, Jasmine test framework is used as the test framework for the API test automation of this thesis. Inside this framework, there are two functions such as `describe` and `it`. The `describe` function contains two parameters. The first parameter is a string, which is often used to set the context of its test file. It only shows what acceptance criteria is defined in its spec. The second parameter is a function, which usually has 1 `it` block or many `it` blocks that depends on how many test cases are designed. In an `it` block, there are two parameters, which are a string and a function. Once again, this first parameter, a string, is used for test case statements in details. Then, the second parameter, a function, lets users use all kinds of functions, which are given by Jasmine test framework. These functions assert what is coming back or what is expected from their JavaScript classes.

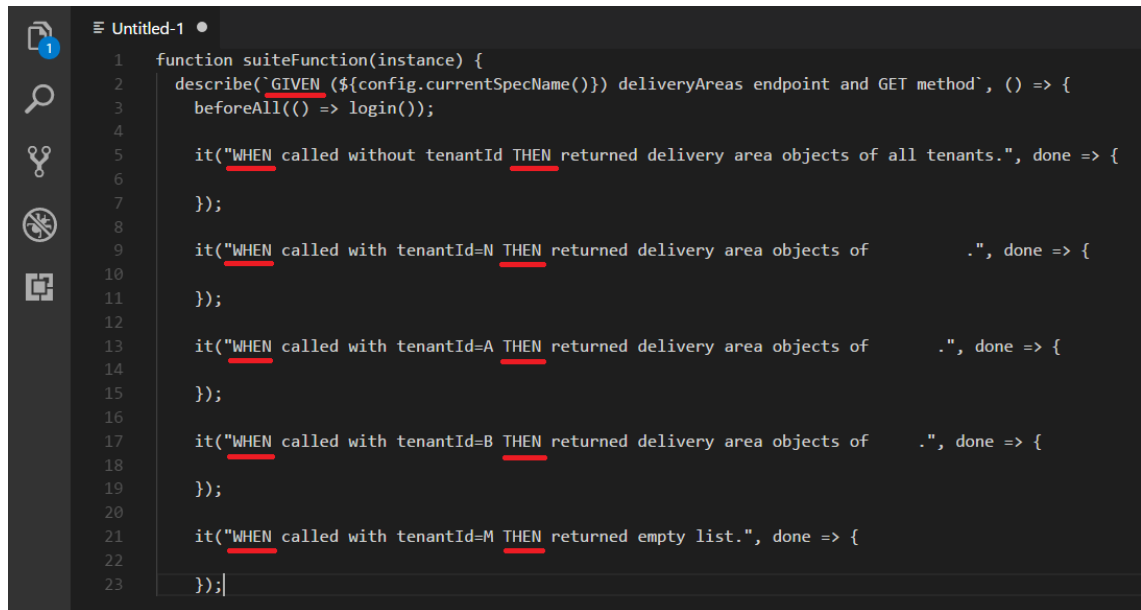
```

describe("A suite", function() {
  it("contains spec with an expectation", function() {
    expect(true).toBe(true);
  });
});
  
```

Figure 15: Jasmine test framework introduction layout (Jasmine n.d.)

Thanks to the BDD feature of Jasmine test framework, test layout is made by the human-readable syntax called `Given`, `When` and `Then`. Jasmine test framework organizes API test

automation of this thesis into three different parts, which are test suite, test case and test step. Test suite uses `Given` syntax. Test case uses `When` syntax. Test step uses `Then` syntax. The test cases and test steps are designed to test different valid and invalid input values when `deliveryareas`' endpoints are called in Area shared service. They are designed as the photo indicates below. The red-highlighted indicator is used to point out `Given`, `When` and `Then` syntax.



```

1  function suiteFunction(instance) {
2  describe("GIVEN (#{config.currentSpecName()}) deliveryAreas endpoint and GET method", () => {
3    beforeAll(() => login());
4
5    it("WHEN called without tenantId THEN returned delivery area objects of all tenants.", done => {
6
7    });
8
9    it("WHEN called with tenantId=N THEN returned delivery area objects of .", done => {
10
11    });
12
13    it("WHEN called with tenantId=A THEN returned delivery area objects of .", done => {
14
15    });
16
17    it("WHEN called with tenantId=B THEN returned delivery area objects of .", done => {
18
19    });
20
21    it("WHEN called with tenantId=M THEN returned empty list.", done => {
22
23    });

```

Figure 16: Test cases are designed to auto-test `deliveryareas`' endpoints

Overall, the test plan and test design, which are carried out in the first sprint, have significant implications to understand how API test automation is implemented in the second sprint. The plan and design are made for tester to be sure what test tools, test framework and programming language are going to support TA. They also help tester prevent the scope and objectives from being ignored or lightly checked. Without test plan and test design, any test automation may result in a lot of activity but little value.

7.2 Sprint 2: Agile test automation implementation

In this section test plan and test design of API test automation are applied and requirement specification is followed. Agile TA implementation of this thesis is conducted sequentially by developing, testing, measuring, releasing, receiving feedback and maintaining. This sprint includes two weeks, of which developing, testing, measuring and releasing happen in the first week whereas receiving feedback and maintaining occur in the second week. With Agile methodology, a set of these practices are utilized through collaboration between self-organizing and team's efforts.

7.2.1 Develop

Based on the workflow of JIRA, before starting its test, the ticket needs to be changed to “In Development” status. API test automation of this thesis is developed to test five different input cases, which usually occur when a service calls to the Area shared service and try to reach the `deliveryareas` endpoint. GET method is used in this TA. This method functions as a request, which tries to get or to achieve the database or information that includes in this endpoint. The five specs, which are five input cases, are written in JavaScript programming language and Jasmine test framework. Hereby they are run locally according to the instruction of the X Test Automation project. The red-highlighted indicator is used to point out the command line that is used to run five specs.

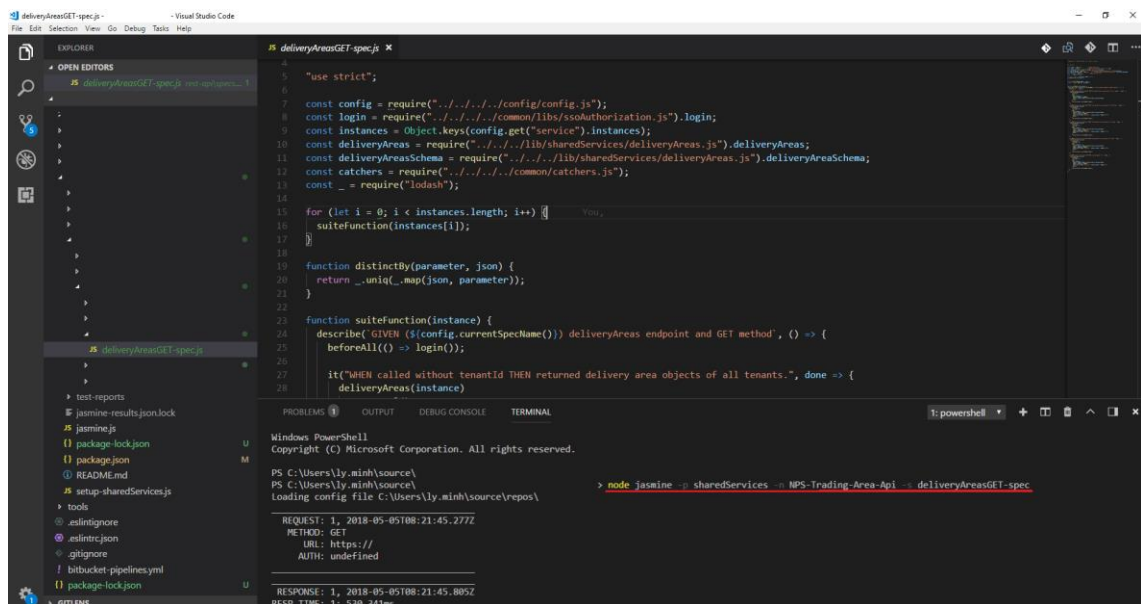


Figure 17: Test cases are run locally

7.2.2 Test and measure

This spec contains various callback methods that keep track of when a test case starts, succeeds or fails. It takes around 3.301 seconds to run this spec, which includes five test cases at the running time. The first case tests when a service calls without a specific `tenantId`, then this endpoint is expected to return all delivery areas of all tenants. The second case tests when a service calls with `tenantId=N`, then this endpoint is expected to return all delivery areas of X Group. Value N is considered for the value of X Group’s `tenantId` abbreviation. The third case tests when a service calls with `tenantId=A`, then this endpoint is expected to return a delivery area of the first partnership of X Group, which is a power exchange company in Europe. Value A is used for the value of its `tenantId` abbreviation. The fourth case

tests when a service calls with `tenantId=B`, then this endpoint is expected to return a delivery area of the second partnership of X Group, an independent European energy exchange company in Europe. Value B is used for the value of its `tenantId` abbreviation. The fifth case tests a service calls when a service calls with `tenantId=M`, a random invalid value, then this endpoint is expected to return a blank space, which has `[]` output on the interface.

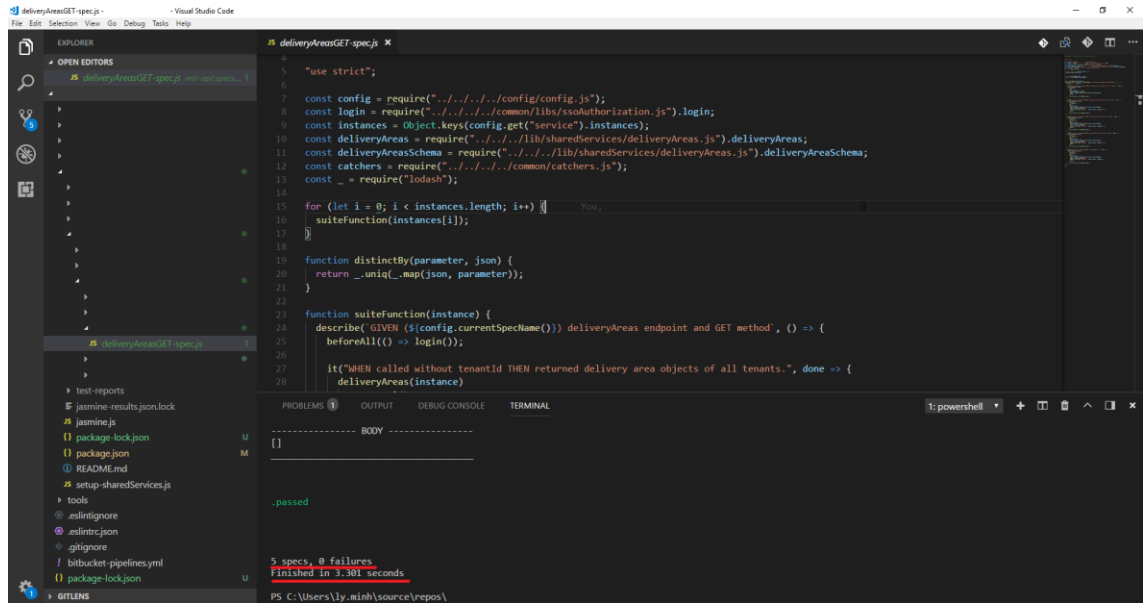


Figure 18: Five specs are executed and time-measured locally

With similar logic and order of the API test automation above, these five cases can also be done manually by Swagger or Postman test tools. A Microsoft desktop clock, which works as a time calculator, measures the consuming time to test these five cases manually. The purpose of this time measurement is to compare how many times that TA works faster than the manual test. For doing this, each case has an input value such as N, A, B and M into the `Value` field of `deliveryareas` endpoint in Swagger. Additionally, there is one case, which tests when inputting no value. So in that case, only hitting “Try it out!” button is done. The red-highlighted indicator is used to point out the input value and the “Try it out!” button of `deliveryareas` endpoint in Swagger.

.AreaService.

Area Show/Hide | List Operations | Expand Operations

GET /api/v1/deliveryareas Provide all delivery areas with given filters. Only active areas are given by default.

Response Class (Status 200)
OK

Model | Example Value

```
[
  {
    "connections": [
      {
        "id": "string",
        "fromAreaId": 0,
        "toAreaId": 0,
        "validFrom": "2018-05-05T18:28:52.248Z",
        "validTo": "2018-05-05T18:28:52.248Z",
        "maximumCapacityKw": 0,
        "referenceId": "string"
      }
    ]
  }
]
```

Response Content Type

Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|------------------------|--------------------------------|-------------|----------------|-----------|
| searchRequest.tenantId | <input type="text" value="N"/> | | query | string |

Response Messages

| HTTP Status Code | Reason | Response Model | Headers |
|------------------|--|----------------|---------|
| 400 | Incoming data are incorrect | | |
| 401 | Login or password are incorrect | | |
| 500 | Exception was thrown during the request processing. | | |
| 502 | Database can not be reached with current configuration | | |
| 504 | Database request timeout | | |

[Try it out!](#) [Hide Response](#)

Curl

```
curl -X GET --header 'Accept: application/json' 'https://'
```

Figure 19: Deliveryareas' endpoint is manually tested on Swagger

Inputting value into a text field, hitting the button and scrolling down to see the response code take three seconds if one case is tested in Swagger. Testing five cases, which requires the tester to input different values, then hit the button and scroll down to see the responses, takes around fifteen seconds in total. That means manual test is slower than TA undoubtedly in this situation. For testing other endpoints, in which the tester must input different values of many different parameters, it is possibly going to take more time. Furthermore, the long-repeated details of inputting different values may result in typographical errors and omission. This consequence probably reduces the efficiency of agile processes.

In view of all that has been mentioned so far, after the API test automation of five specs is tested successfully, they are committed, pushed and merged into the `master` branch. Based on the workflow of JIRA, after being merged, the ticket needs to be changed from "In Development" status to "Ready for Testing" status. As being planned, the test file is named with

endpoint URL + method format so that it has the name of `deliverAreasGET-spec.js`. Additionally, as being planned, it is stored in the `common` folder of X Test Automation project. The red-highlighted indicator is used to point out the right location, where API test automation of `deliveryAreasGET-spec.js` file is merged into the `master` branch of the project.

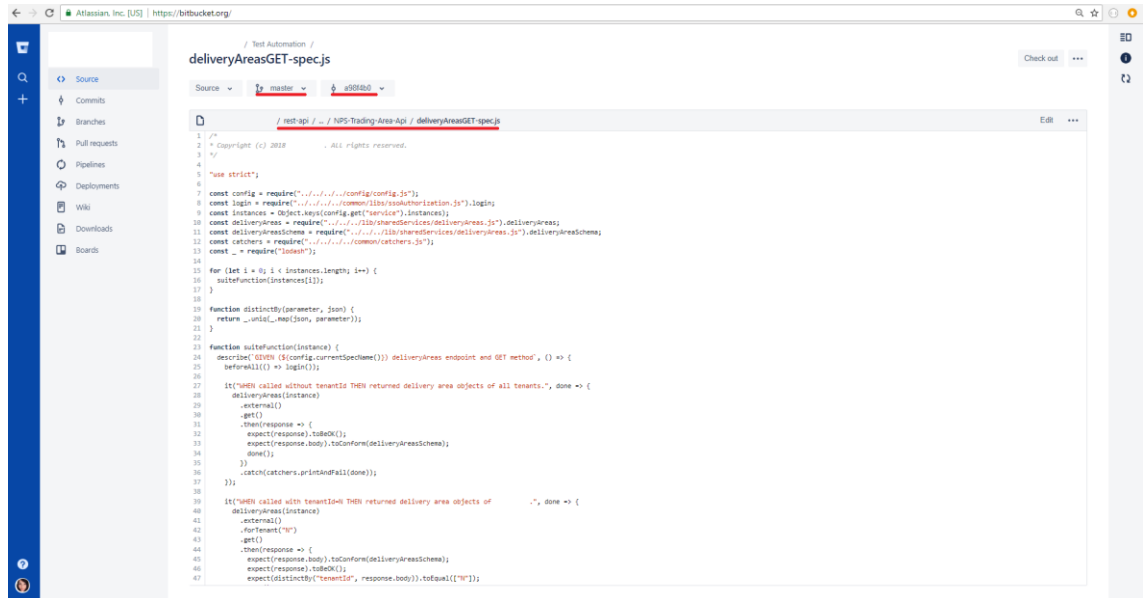


Figure 20: API test cases are merged into master branch of X Test Automation project

7.2.3 Release

So far the Agile TA implementation has focused on running test cases and storing the spec file. The following section discusses the release phase of this API test automation. At the time these five specs are created, there has been no plan for the whole X Test Automation project to be deployed in the Pre-Production and Production environments yet. The five specs written for this thesis topic are implemented into the Quality Assurance environment after being merged into the `master` branch. The specs do not need to pass the Development environment, but they are pushed straight to Quality Assurance environment, which stands after the Development environment. The reason is that these specs are checked and merged by other QA engineer, whereas QA team can provision and track back these specs in their working environment that is QA environment.

Up to now, these specs are not in the release of Production. They are a small contribution for the TA of shared service Area in the X Test Automation project. This project has not had any release in Pre-Production and Production yet. It has been planned that X is building the TA system that the QA engineers of all teams can develop API test automation and end-to-end UI test automation for the X Test Automation project. It is hoped that the TA system is going to

have a careful and stable strategy. That means the TA system must have all basic processes of Agile such as Plan, Design, Develop, Test, Release, Feedback and Maintenance to be in the business party and technical party's controls. So that the release of X Test Automation project can be possible in the Production environment.

7.2.4 Feedback

In the case of feedback, it is given after this implement and deployment. It is the feedback of QA engineer, who is the supervisor of this thesis at work placement. The supervisor checks and agrees with the contribution of this API test automation, then inputs the `Copyright © 2018 X. All rights reserved. comment inside the specs on the system.`

7.2.5 Maintain and document

With regard to the maintenance after deployment, it is observed that the specs have not caused any systematic problems. Before being deployed in QA environment, there is no message in error queues. API test automation of this thesis is required to inform the relevant parties, who are QA team and DevOps team after TA deployment is completed in Octopus. The further maintenance of X Test Automation project, where the specs locate, is going to be taken by QA team and DevOps team together. Both teams are free to define and agree on a set of project maintenance in the future. Finally, on Confluence, a multimedia content storage, the passed test cases are documented in an API Forward Backward Traceability log sheet. It is a mapping requirement whereas test cases and the required values of all API endpoints should be tested.

8. Evaluation

The thesis contributes test automation for one API endpoint, which is `deliveryareas` and its method, which is `GET` method. Based on the workflow of JIRA, after being tested by QA team and product owner, the ticket has been verified and changed from “Ready for Testing” status to “Completed” status. The thesis outcome has ensured and completed the scope and objectives, which were set up at the beginning.

The thesis has successfully reached the five goals, which were defined in the requirement specification. With the first goal, API test automation of thesis has increased the testing speed to deliver fast and qualified feedback. It was proved when applying time measurement method to compare the speed of test automation and manual test. It took from two to three seconds to test five specs, which were five different test cases, by conducting test automation in Visual Studio Code application. Likewise, it took from twelve to fifteen seconds to test the same logic and procedures of those five test cases manually in Swagger application. Therefore, API test automation has delivered feedback faster than manual test does. API test automation has also avoided the iterating test actions, which can cause human’s typographical errors and omission.

With the second goal, API test automation of this thesis has given confidence that passed tests have been deployed and maintained in QA environment. It is on the way to Production deployment when a whole TA system is built properly as X’s plan. With the third goal, until now regardless of feature’s changes, the passed tests ensure its reliability and stability. With the fourth goal, the passed tests have followed all requirement specification, X’s Acceptance Criteria and API test rules. It has been proved when API test automation of Area service has passed test results, and the passed test has been licensed. Finally, with the fifth goal, API test automation of thesis has been designed and documented in a shared repository due to the structural steps of TE. The TE was defined, built and validated thanks to the multiple functionalities of developing technologies. They are JIRA, Visual Studio Code, Bitbucket, Bamboo, Octopus and Confluence applications as well as the JavaScript programming language and Jasmine test framework.

9. Conclusion

The thesis has had its scope and objectives carried out successfully. The thesis applied TA in writing scripts to test one API endpoint of a shared service, Area, for X. The passed tests have been verified, licensed and stored in the test automation project of the company. Furthermore, the thesis has defined, built and validated its testing environment by researching, studying and applying new technologies that X has been using. Moreover, the thesis practices were conducted in TA goals, API test automation rules, deployment instructions and ethical guidelines of the company and self-reflection.

The present study lays the groundwork for future research in API test automation of other services in X Test Automation project. There is still more than one endpoint in Area service need to be planned, designed and tested. However, the question raised by this study is what should be planned and how to proceed the plans to build X Test Automation project until the day X has a unified and centralized API test automation and end-to-end UI test automation. There are more than ten shared services, which support mutually and support the other services such as Intraday Continuous Trading, Market Data, Compliance as Urgent Market Message, Auction and Single Sign On. It is discussed and agreed in X Group's business process that the succession planning of any projects should contain many procedures in their decisive phases such as identification of key roles and contributors, successor identification, career plan and talent management. To achieve this goal, which makes X Test Automation project become an enriched, reliable and stable system, the X Group is suggested to have a greater focus on this succession planning.

References

Public printed sources

Kaner, C., Bach, J. & Pettichord, B. 2002. Lessons Learned in Software Testing. New York: Wiley Computer Publishing.

Electronic sources

Agile Model & Methodology: Guide for Developers and Testers. No date. Guru99. Accessed 31 March 2018.

<https://www.guru99.com/agile-scrum-extreme-testing.html>

AUTOMATION TESTING Tutorial: Process, Planning & Tools. No date. Guru99. Accessed 31 March 2018. <https://www.guru99.com/automation-testing.html>

Ben. 2017. JavaScript unit testing frameworks: Comparing Jasmine, Mocha, AVA, Tape and Jest. RAYGUN, 25 May. Accessed 22 April 2018. <https://raygun.com/blog/javascript-unit-testing-frameworks/>

Best agile methodology books for project development and management. 2018. Online-BooksReview. Accessed 31 March 2018. <https://www.onlinebooksreview.com/articles/best-agile-books>

Cobb, G. 2014. Writing Beautiful Specs with Jasmine Custom Matchers. Pivotal, 12 February. Accessed 15 April 2018. <https://content.pivotal.io/blog/writing-beautiful-specs-with-jasmine-custom-matchers>

Custome_matcher.js. No date. Jasmine. Accessed 15 April 2018. https://jasmine.github.io/2.0/custom_matcher.html

Eriksson, U. 2013. The Most Important Testing Tool is the Software Tester's Brain. ReQtest, 08 March. Accessed 08 April 2018. <https://reqtest.com/testing-blog/the-most-important-testing-tool-is-the-software-testers-brain/>

Ethical Hacking career options for qa professionals. No date. Software Quality Assurance & Testing. Accessed 15 April 2018. <https://sqa.stackexchange.com/questions/24340/ethical-hacking-career-options-for-qa-professionals>

Garg, P. 2017. Agile Methodology: Why Is It Important For Your Start-up?. Openxcell, 03 May. Accessed 31 March 2018. <https://www.openxcell.com/agile-methodology-important-start>

Git that grows with you. No date. Bitbucket. Accessed 22 April 2018. <https://www.atlassian.com/software/bitbucket/enterprise/data-center>

GT Partner Team. 2017. 12 Reasons Why Developers Should Use Bitbucket. GuidingTech, 17 January. Accessed 01 April 2018. <https://www.guidingtech.com/63369/use-bitbucket/>

Hickerson, M. 2016. The Agile Bicycle: A Better Analogy for Software Development. Dotdev, 15 May. Accessed 31 March 2018. <https://m.dotdev.co/the-agile-bicycle-829a83b18e7>

Horne, G. 2014. A (Very) Brief History of Test Automation. LinkedIn, 07 October. Accessed 08 April 2018. <https://www.linkedin.com/pulse/20141007123253-16089094-a-very-brief-history-of-test-automation/>

Intro to the Postman API. No date. Postman. Accessed 01 April 2018. https://www.getpostman.com/docs/v6/postman/postman_api/intro_api

Jasmine Behavior-Driven JavaScript. No date. Jasmine. Accessed 22 April 2018. <https://jasmine.github.io/>

Jenkov, J. 2017. Java Regex - Matcher. Tutorials.Jenkov, 06 November. Accessed 15 April 2018. <http://tutorials.jenkov.com/java-regex/matcher.html>

JSON vs XML. No date. W3Schools. Accessed 01 April 2018. https://www.w3schools.com/js/js_json_xml.asp

Keough, B. 2017. The 6 best Confluence pages for building a DevOps culture. Atlassian Blog, 21 August. Accessed 22 April 2018. <https://www.atlassian.com/blog/confluence/best-confluence-pages-for-devops-culture>

Lesunova, I. 2018. 5 New Bitbucket Apps to Keep You More Organized - Fourth Quarter 2017. StiltSoft, 16 February. Accessed 22 April 2018. <https://stiltsoft.com/blog/2018/02/5-new-bitbucket-apps-to-keep-you-more-organized-fourth-quarter-2017/>

McPeak, A. 2018. The Criteria to Consider for Choosing JavaScript Testing Frameworks. Cross-BrowserTesting, 18 January. Accessed 08 April 2018. <https://crossbrowsertesting.com/blog/testing-frameworks/javascript-testing-frameworks/>

Nielsen, J. 2001. Error Message Guidelines. Nielsen Norman Group, 24 June. Accessed 01 April 2018. <https://www.nngroup.com/articles/error-message-guidelines/>

Northcutt, S. No date. Security Laboratory: Methods of Attack Series. SANS Technnology Institute. Accessed 01 April 2018. <https://www.sans.edu/cyber-research/security-laboratory/article/race-cndtns>

Octopus Deploy Documentation. No date. Octopus. Accessed 22 April 2018. <https://octopus.com/docs/getting-started>

Power-Morse, A. 2016. JavaScript Frameworks: 5 Front and Back-End Options We Love. Airbrake, 16 November. Accessed 08 April 2018. <https://airbrake.io/blog/javascript/javascript-frameworks-love>

Reddy, M. 2015. What is TestNG?. Built.io, 08 June. Accessed 31 March 2018. <https://www.built.io/blog/what-is-testng>

Renaudin, J. 2016. Better Test Automation, Metrics, and Measurement: An Interview with Mike Sowers. Stickyinds, 06 May. Accessed 31 March 2018. <https://www.stickyinds.com/interview/better-test-automation-metrics-and-measurement-interview-mike-sowers>

Rosenstock, L. 2017. 5 Reasons You Should Use OpenAPI/ Swagger for your APIs. BlazeMeter, 10 October. Accessed 22 April 2018. <https://www.blazemeter.com/blog/five-reasons-you-should-use-openapi-swagger-for-your-apis>

Running Tests in Parallel. No date. xUnit.net. Accessed 01 April 2018. <https://xunit.github.io/docs/running-tests-in-parallel#background>

Selenium WebDriver. 2018. SeleniumHQ. Accessed 22 April 2018. https://www.seleniumhq.org/docs/03_webdriver.jsp

SOAP vs REST 101: Understand The Differences. No date. SoapUI. Accessed 01 April 2018. <https://www.soapui.org/resources/api-testing/article/soap-vs-rest-api.html>

Software Engineering code of Ethics. IEEE-CS/ACM joint Take Force on Software Engineering Ethics and Professional Practices. No date. IEEE Computer Society. Accessed 15 April 2018. <https://www.computer.org/web/education/code-of-ethics>

Stahl, M. 2018. Ethics in Software Testing. Stickyminds, 05 March. Accessed 15 April 2018.
<https://www.stickyminds.com/article/ethics-software-testing>

State of API Security. No date. SoapUI. Accessed 01 April 2018.
<https://www.soapui.org/learn/security/state-of-api-security.html>

Swagger-The World's Most popular API Tooling. No date. Swagger. Accessed 01 April 2018.
<https://swagger.io>

Testing REST API Manually. No date. Guru99. Accessed 01 April 2018.
<https://www.guru99.com/testing-rest-api-manually.html>

Top 15 UI Test Automation Best Practices You Should Follow. 2017. BlazeMeter, 14 November. Accessed 01 April 2018. <https://www.blazemeter.com/blog/top-15-ui-test-automation-best-practices-you-should-follow>

Top 20 API Testing Tools In 2018: REST & SOAP. No date. Guru99. Accessed 08 April 2018.
<https://www.guru99.com/top-6-api-testing-tool.html>

Top Automation Testing Frameworks Professionals Love. 2017. CrossBrowserTesting, 28 March. Accessed 15 April 2018. <https://crossbrowsertesting.com/blog/test-automation/top-automation-frameworks-testers/>

Understanding the Bamboo CI Server. 2017. Bamboo Support, 20 July. Accessed 22 April 2018.
<https://confluence.atlassian.com/bamboo/understanding-the-bamboo-ci-server-289277285.html>

UNIT Testing Tutorial - Learn in 10 Minutes. No date. Guru99. Accessed 01 April 2018.
<https://www.guru99.com/unit-testing-guide.html>

Using JIRA for Test Case Management (Challenges, Pros/Cons, Tips & More!). No date. QASymphony. Accessed 22 April 2018. <https://www.qasymphony.com/blog/guide-using-jira-test-case-management/#>

Vasudevan, K. 2017. Design First or Code First. Swaggerhub, 21 February. Accessed 22 April 2018. <https://swaggerhub.com/blog/api-design/design-first-or-code-first-api-development/>

Wagner, J. 2014. Review: Postman Client Makes RESTful API Exploration a Breeze. ProgrammableWeb, 27 January. Accessed 22 April 2018.

<https://www.programmableweb.com/news/review-postman-client-makes-restful-api-exploration-breeze/brief/2014/01/27>

What is Load Testing?. No date. SmartBear. Accessed 01 April 2018.

<https://smartbear.com/learn/performance-testing/what-is-load-testing/>

Zalenum. No date. Zalando. Accessed 08 April 2018.

<https://opensource.zalando.com/zalenum/>

Unpublished sources

Foppa, J. & Pehkonen, T. 2017. SSO in X Solutions. X Mentoraid, 13 December. Accessed 22 April 2018.

X's Ethical guideline. 2018. X Finland Oy. Accessed 15 April 2018.

X XBID Projects. 2014. X's XBID. Accessed 22 April 2018.

Figures

| | |
|---|----|
| Figure 1: How Agile methodology is illustrated | 9 |
| Figure 2: Project management in Agile | 10 |
| Figure 3: Agile software testing life cycle..... | 11 |
| Figure 4: XML format example..... | 12 |
| Figure 5: JSON format example | 13 |
| Figure 6: Testing environment workflow | 25 |
| Figure 7: Test automation request on JIRA system | 30 |
| Figure 8: JIRA workflow | 30 |
| Figure 9: Test Automation project on Bitbucket..... | 31 |
| Figure 10: API endpoints of Area service to be tested on Swagger | 31 |
| Figure 11: First API endpoint of Area service is checked by Postman tool | 32 |
| Figure 12: Second API endpoint of Area service is checked by Postman tool | 32 |
| Figure 13: Token access flow | 33 |
| Figure 14: One delivery area is assigned to one market area..... | 35 |
| Figure 15: Jasmine test framework introduction layout | 35 |
| Figure 16: Test cases are designed to auto-test deliveryareas' endpoints | 36 |
| Figure 17: Test cases are run locally | 37 |
| Figure 18: Five specs are executed and time-measured locally | 38 |
| Figure 19: Deliveryareas' endpoint is manually tested on Swagger | 39 |
| Figure 20: API test cases are merged into master branch of X Test Automation project..... | 40 |

Captions

| | |
|--|----|
| Caption 1: Using custom matcher in API test automation | 21 |
| Caption 2: Predefined data in database | 21 |
| Caption 3: Example of called method iteration in API test automation | 22 |
| Caption 4: Example of Builder pattern in API test automation | 23 |