



Aalborg University Copenhagen

A.C. Meyers Vænge 15
2450 København SV

Secretary: Maiken Keller

AALBORG UNIVERSITY

STUDENT REPORT

Semester:

ICTE 4

Title:

Text-to-Test-Mapper

Project Period:

Spring Semester 2019

Semester Theme:

Advanced ICT systems

Supervisor(s):

Per Lynggaard

Project group no.:

4.1

Member(s):

20137201 Edgars Raudive
20147325 Marius Ukrinas

Pages: 105

Finished:

June 5, 2019

Abstract:

The aim of the report was to define a solution that is able to utilize machine learning in combination with a web application to automate Behaviour Driven Development based Test Case creation in an autonomous manner. A combination of ReactJS web application, Node.js server, MongoDB and an LSTM neural network model provides users a way to converge their custom Test Case definitions into Test Cases consisting of predetermined Test Step formulations. This is achieved by performing semantic comparison between user inputs and the predetermined Test Steps. The resulting system is addressed by Text-to-Test-Mapper and is designed from the perspective of the project's case study company - GN ReSound.

All group member are collectively responsible for the content of the project report. Furthermore, each group member is liable for that there is no plagiarism in the report. Remember to accept the statement of truth when uploading the project to Digital Exam

Text to Test Mapper (TTTM)



Group number: 4.1

Contents

1	Introduction	1
1.1	Project motivation	2
1.2	Research question	4
1.3	Delimitations	4
1.4	Expected Outcome	4
2	Methodology	6
2.1	Process Model	6
2.2	Primary research	8
2.2.1	Preparation of the data set for Machine learning	9
2.2.2	Low-Fi prototype user testing	9
2.3	Secondary research	10
2.4	Implementation independent model and Requirements	10
2.5	Testing	11
2.5.1	Machine Learning Testing	11
2.5.2	Black box Software testing	11
2.5.3	User-Testing	12
3	State of the Art	13
3.1	Project's development premise	13
3.2	Existing solutions	14
3.2.1	TextRazor	15
3.2.2	Aylien	16
3.2.3	Dandelion	16
3.3	Machine Learning Algorithms	17
3.3.1	Machine Learning Overview	18
3.3.2	Latent Dirichlet Allocation (LDA)	18
3.3.3	Latent Semantic Analysis - Singular Value Decomposition (LSA-SVD)	19
3.3.4	Hidden Markov Model (HMM)	20
3.4	Neural networks	20
3.4.1	CNN	22

3.4.2	RNN	22
3.5	Machine learning tools	24
3.5.1	TensorFlow	24
3.5.2	PyTorch	24
3.5.3	Microsoft Cognitive Toolkit	25
3.5.4	Keras	25
3.6	Web service frameworks	26
4	Analysis	28
4.1	Background of Analysis	29
4.2	Web stack comparison	30
4.3	Machine Learning vs Neural Networks	32
4.3.1	Analysis of Machine Learning algorithms	33
4.3.2	Neural Network analysis	35
4.4	Tools for Machine Learning	41
4.5	Preparation of the data set for Machine learning	42
4.6	Low-Fi prototype user testing	44
4.7	Requirement specification	46
4.7.1	User available actions in TTTM	46
4.7.2	The main information flow of the TTTM system	48
4.7.3	Requirements	50
5	System design	55
5.1	Solution overview	55
5.2	System functionality flows	59
5.2.1	Login	59
5.2.2	Add Test Case	61
5.2.3	Edit Test Case	63
5.2.4	Delete Test Case	63
5.3	API Endpoints	64
6	Implementation	66
6.1	Data structure	67
6.1.1	Test Steps collection	67
6.1.2	User collection	68
6.1.3	Test Cases collection	69
6.2	Client-side	70
6.2.1	Managing application data	71
6.2.2	API Calls	73
6.2.3	Routing of application	74
6.2.4	Client-side GUI	75
6.3	Back-end	76

6.3.1	Machine Learning algorithm	77
6.3.2	Server-side	80
7	Testing	87
7.1	Machine Learning Accuracy Testing	87
7.2	Black Box Software testing	89
7.3	User testing	90
8	Reflections	92
8.1	Evaluation of project related processes	92
8.2	Complete solution vs prototype	94
8.3	Future improvements	95
8.3.1	Dynamic variable detection	95
8.3.2	Test Step addition by users	95
8.3.3	Potential perspectives of the TTTM solution	96
9	Conclusion	97
	Bibliography	99
	Appendix - can be found in the attached document	

Acronym	Description	Acronym	Description
API	Application Programming Interface	MERN	Mongo, ExpressJS, ReactJS, NodeJS
AWS	Amazon Web Services	ML	Machine Learning
BDD	Behavior Driven Development	MoSCoW	Must, Should, Could, Won't
BRNN	Bidirectional Recurrent Neural Network	MVC	Model, View, Controller
CMS	Content Management Systems	NFR	Non-Functional Requirements
CNN	Convolution Neural Network	NLP	Natural Language Processing
CNN	Convolution Neural Network	NN	Neural Network
CNTK	Cognitive Toolkit	OS	Operating System
CPU	Central Processing Unit	PHP	Hypertext Preprocessor
CRUD	Create, Read, Update and Delete	PI	Project Increments
CSS	Cascade Style Sheets	QA	Quality Assurance
DAN	Deep Adversarial Networks	RAD	Rapid Application Development
DNN	Deep Neural Networks	RDBMS	Relational Database Management System
DOM	Document Object Model	ReLU	Rectified Linear Unit
FFNN	Feedforward Neural Network	R&D	Research and Development
FR	Functional Requirements	RNN	Recurrent Neural Networks
GPU	Graphics Processing Unit	SAFe	Scaled Agile Framework
GRU	Gated Recurrent Units	SPA	Single Page Application
GUI	Graphical User Interface	SQL	Structured Query Language
HI	Hearing Instrument	SVD	Singular Value Decomposition
HMM	Hidden Markov model	TC	Test Case
HTML	Hypertext Markup Language	TD	Temporal Difference
IEEE	Institute of Electrical and Electronics Engineers	TDD	Test Driven Development
IoT	Internet of Things	TS	Test Step
IT	Information Technology	TTM	Text-to-Test-Maper
JS	JavaScript	UI	User Interface
JSON	JavaScript Object Notation	UML	Unified Modeling Language
JSX	Java Script XML	URI	Uniform Resource Identifier
LAMP	Linux, Apache, MySQL, PHP	URL	Uniform Resource Locator
LDA	Latent Dirichlet Allocation	USD	US Dollar
LSA	Latent Semantic Analysis	UX	User experience design
MEAN	Mongo, ExpressJS, AngularJS, NodeJS		

Chapter 1

Introduction

Nowadays software development is an ever-growing industry which is supported by the fact that in the start of 2018 IT products were 14% of US overall export, losing only to Transportation and motor vehicle industry (19%) [1]. The total revenue of IT product exports reached 202 billion USD, and the industry expected growth of at least 5% [1]. A large part of Technology development is the Quality assurance (QA) process that decides if the product is ready for being deployed in the market or not. This process faces a lot of different challenges, especially in terms of test automation. Based on the World Quality Report of 2018-2019 [2] it can be said that only 14-18% of test activities are automated which implies that the remaining percentage of testing is deemed costly in terms of the resources companies need to spend for their product verification and validation. However, how can the QA process be improved without major rehiring and retraining of the company's staff?

This report aims to answer this question using an automated software system. The system would be able to benefit from the already existing documentation the companies have for their test cases. The basic idea behind the system is to help companies increase automated and decrease manual testing. This approach attempts to fill the communication gap between "code people" and "requirement people".

The following chapters introduce the current background of product verification and automated testing in Agile environment together with the motivation for this project which reveals the reasoning behind the choice of the use case of GN ReSound and the current problems this company faces. The chapter follows with the research question of the project supported by delimitations that define the scope of the report. Afterwards, the description of the expected outcome product is introduced.

1.1 Project motivation

It is a common practice in the industry to develop systems based on a set of requirements created beforehand. Since 1998 there exists an IEEE 830 standard [3] that provides a template for developing product requirements. As described in the standard, these requirements create a baseline for validation and verification of the product. The purpose of those processes is to ensure that the product's requirements and specifications are met and that it can serve its designed purpose [4]. The current trends in quality assurance departments suggest that companies lean towards Agile process models for verifying these requirements [5].

One of the approaches for requirement verification is using different kinds of testing. In Agile environments where it is important to have continuous improvement of the product, it is crucial to have iterative testing phases. These phases can include both manual and automated testing. Manual testing includes such testing types as exploratory and usability tests [6]. On the other hand, it is much more common to do unit, integration, Graphical User Interface (GUI) and regression tests in an automated manner [7] as they deal with repetitive tasks that verify the system's overall stability. Unit and integration tests verify the code stability on the developers' end and GUI together with regression tests are usually performed by the QA department employees. The current trends [2] also suggest that there is considerable interest in new ways of creating the automated test cases and a stable interest throughout the recent years in Machine Learning (ML) in QA and test design automation shown in Figure 1.1.

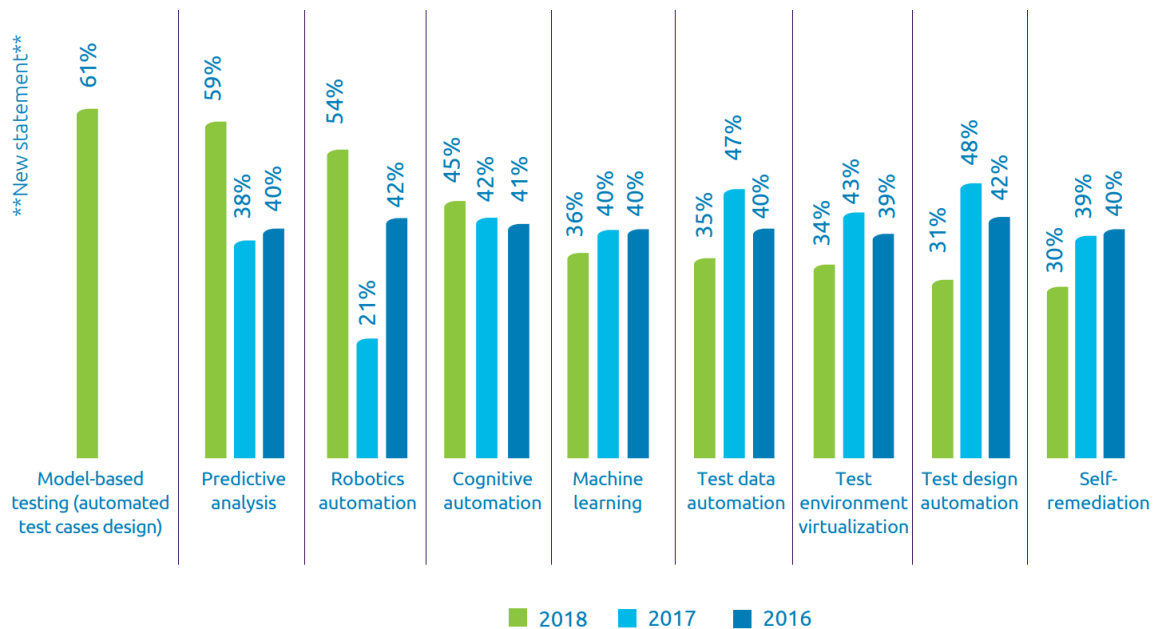


Figure 1.1: Projected business interest in automation techniques [2]

There is a multitude of ways to implement and document both manual and automated tests, which usually are company specific. A company named GN ReSound was chosen as a case study for this project. This company is one of the participants in the market of the hearing aid industry. The company strives to innovate in its specialty field with new hearing aids that users can control from their smartphones. GN ReSound has been in the hearing instrument business since 1943 [8], so it is no surprise that the R&D department had applied Waterfall methodologies for a long time. The company's R&D department focuses on Hearing instrument (HI) firmware, Fitting Software and smartphone application development. Only recently, GN ReSound has switched to a more Agile approach for developing their products called SAFe [9]. This methodology divides the development process of the whole company into Project Increments (PI) which in essence are three month long iterations after which all of the teams have a PI meeting to establish goals and dependencies for the next PI. The goal functionalities for each team are split into multiple "features" and whenever one is deployed the whole system is tested against requirements.

Just as many other companies GN ReSound uses both manual and automated testing, focusing more on manual verification testing. However, with the regular shipping of the features it became a challenge for the test teams to do regression and GUI tests for the multitude of product combinations, specifically, different HIs and smartphone applications. At the end of 2018, the test teams started to implement an automation test framework based on Behavior Driven Development (BDD) [10], however, on the contrary to the current testing trends [2] GN ReSound was not delving into solutions based on Machine Learning. The goal of the resulting framework was to help testers in creation of step-based test scripts which are both human and machine readable.

In BDD test cases are called scenarios and are grouped in features by their test purposes. The specific steps in this feature are defined in the framework, and a code script is assigned to each of these steps. Currently, GN ReSound implemented this approach for smartphone application testing. For better viewability, these features and scenarios are also used in the documentation without changing their formulation. This way, the test cases can be easily executed in their Test Framework in an autonomous way. It is important to note that all of the official software development in GN is done using C# language.

In GN ReSound test case creation is initiated by creating its documentation first. The problem that arises from this approach is that the test case or scenario creator needs to have the full knowledge of all the test steps defined in the source code of the Automation Test Framework to know what kind of test cases they can create. Moreover, this approach is not very scalable, as upon creating new step definitions in the automated framework, the test case creators will need to add new steps to their "dictionary" of allowed test steps. Such method requires a supporting system to assist the users, which can be defined by the problem statement of this project:

1.2 Research question

How can test case documentation be used to create test scripts which can be executed autonomously?

1. How can machine learning algorithms be utilized to support the dynamic creation of test cases?
2. How can a web application help users in the process of generating test cases?
3. How can such a system be adopted in the context of the selected case study - GN ReSound?

1.3 Delimitations

- The project will not focus on the usability and the design of the Client-side of the provided web application.
- For the prototype of this project, only 16 documented test steps provided by the case study company were used, see Appendix D.
- The project's prototype will be designed only with one type of users - test case Manager in mind.
- The system will be designed to support test case and test step management only in relation to the GN ReSound application - Smart3D.

1.4 Expected Outcome

The expected outcome of the project is represented in Figure 1.2. It is inspired by the research question of the project and the preliminary design thoughts of the group members. The solution will hereby be referred to as Text-to-Test-Mapper or TTTM.

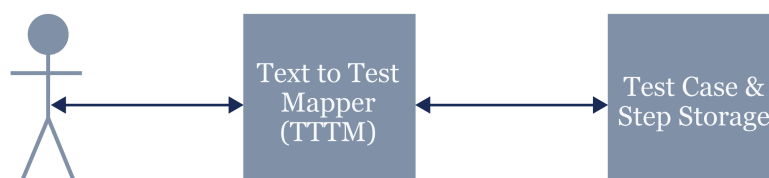


Figure 1.2: Expected outcome diagram

The user interacts with TTTM system to create, edit and view the test cases. These test cases are verified and stored for efficient user access. The test case generation process in

the TTTM system is connected to the case study company's test step documentation that verifies the validity of the test case. The workflow of the expected outcome is described in a scenario as follows:

Juan is a QA developer in his company. He is outstanding at critical thinking and precise at producing the documentation. His department works with application testing, and Juan usually is the one to create the test cases. Normally after he creates test cases, a team of developers take them and create a code version in their test framework. However, recently the company overwent a change in management and the test framework structure. Now Juan needs to write the test cases in the form of predefined steps, on the contrary to his usual free speech style. The company did implement a support system that helps Juan, who has never seen the code for the framework, to create new valid test cases. After a brief introduction to this system, Juan tried out to write a test case. He wrote a test case in a provided GUI the same way he wrote them before, using his favored formulations. After pressing the "Submit" button, unknown to Juan, the system verified his steps against the ones that the test framework already had. Afterwards, Juan received a number of suggestions to each of his step formulations. After selecting the ones that in Juan's mind were providing the same functionality he wanted, he pressed the "Save" button. And just as that new test case was created without Juan actually knowing the technicalities of this system and the test framework.

Chapter 2

Methodology

In this chapter, a systematic overview of the methodological processes and methods that were used in this project will be presented. The introduced information delves profoundly into the project and is related to the analysis of the report and the development of the prototype. It is described in greater detail in the process model, primary research, secondary research and requirements generation.

2.1 Process Model

At the start of the project, the process model was needed to specify the details of the research and development flow and group work management. As the requirements of the project were not defined at the start of the development process, it was decided to use an iterative approach of their derivation. Therefore, Agile methodologies were researched with the aim to provide flexibility to the project. At the end of the Agile methodology research, Scrum [11] and Rapid Application Development (RAD) [12] methods were considered to be used in the project.

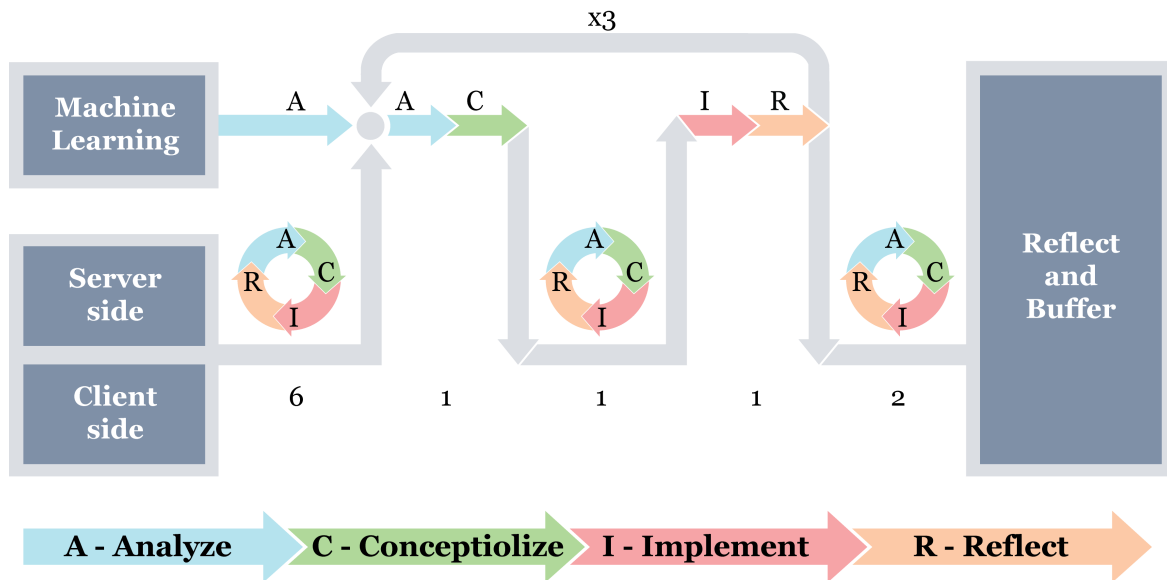
After comparing both, it was clear that the RAD method had a weaker structure in terms of small project needs. It was discovered that RAD has little compatibility with frequent project development decision change. Additionally, it is strongly relying on customers' feedback [12], which was not the main driver of this project. The image of this project development can be described with frequent structural changes, content review and editing. Based on these observations, it was decided to use the Scrum methodology, which proved to be more compatible for a research based project.

Furthermore, the selected Scrum model was modified based on the project needs, that were not only associated with the software but also the report development. Each sprint was a week long iteration. As for group work, the team had regular meetings either in person or remotely. For project task control, a web application called Trello [13] was used.

In Figure 2.1 below, the process model of the project is presented. It shows that the project was divided into two main flows - one for Machine Learning and one for the Web

service consisting of the Server-side and Client-side. Four types of actions are defined, where Analyze and Conceptualize indicate the writing of the report, and Implement and Reflect indicate the actual implementation of the prototype and the discussion of the current state of the project.

The sprint amount for each iteration in Figure 2.1 is identified with numbers. The first six sprints were designed to delve into analysis and development of the Machine Learning algorithms, and preparation of the essentials for the Web Service. As seen in the figure, these essentials were analyzed, conceptualized, implemented and reviewed. After these six sprints, the iterative process of the project's development started. In this process, the first sprint delved into analyzing and learning about Machine Learning, and conceptualizing a possible solution. Next sprint was utilizing the Machine Learning concepts to update the Web Service for compatibility. The last sprint utilizes the prepared Web Service as a framework to implement the conceptualized Machine Learning algorithm and test its performance. This three spring iteration was done three times to gather additional requirements and iterate on the final prototype. Afterwards, two sprints were devoted to report wrap-up and system testing. Lastly, three sprints were used as an overall Reflect phase together with a buffer for potential delays.



In addition, the Scrum method had the following roles and flow steps:

Scrum roles:

- **Product owner** – a team member who was affiliated with the case study company was keeping track of the whole project development and requirement prioritization.
- **Development team** – a whole project group that was responsible for project delivery on time.
- **Scrum master** – an allocated supervisor who kept track of progress in weekly sprints.

Scrum iteration flow:

- **Sprint planning** – At the beginning of each week, a new planning iteration was initialized where decisions about the project's weekly sprint were defined. Each planning session delved into weekly development goals that should be achieved regardless if its prototyping or research.
- **Sprints** – Each sprint intended to limit the time frame for executing the defined tasks and creating new ones.
- **Sprint review** – The sprint's progress was evaluated daily, where team members discussed the status and progression of the tasks.
- **Reflect** – At the end of each week, the reflection on the previously conducted sprint was performed. Based on the reflections, the task was either done or not. In case of failure in completing the task on time, the task was moved to the future sprint iteration.

Furthermore, as a part of the full project planning, a Gantt Chart was used to identify the whole time frame of the project with tasks related to report and prototype development.

2.2 Primary research

The project utilized two distinctive user involvement stages as the sources for primary research. The first stage was a workshop for Test Step collection. It was used to populate the data set of the machine learning part of the project. The inputs for the data set were collected from the context experts as it could increase the accuracy of the machine learning algorithm. The second stage was a user interview with a Low-Fi prototype test with the intention of adding user requirements to the solution. The Product owner of the project would then evaluate which of these requirements are valid for the project scope.

2.2.1 Preparation of the data set for Machine learning

First, this project required textual input that could be utilized for Test Case creation in the system's prototype. The case study company of the project - GN ReSound provided a set of official Test Steps for this purpose. However, for better machine learning performance it was decided to collect user defined Test Step interpretations from experienced users of the GN ReSound application called Smart3D [14]. It was decided to create 5 test goals that these users needed to achieve. While performing these tasks, they were required to use the Think Aloud [15] method. Their inputs were recorded on an audio recorder. This allowed to extract their interpretations and use them for prolonging the data set for the Machine learning part of the solution, as mentioned in the project's research question in Chapter 1.2. The test goals presented to users are defined as follows:

- Observe the Hearing Instrument connectivity
- Read the full About information of the ReSound App
- Navigate to three different programs in three different ways
- Change individual HI volume and mute the program
- Change Bass level to -4 and look at tinnitus menu

The specific test goals were chosen based on 16 Test Steps that GN Resound allowed for publication presented in Appendix D. These five test goals were designed so that all of the available Test Steps could be covered in a natural flow of actions.

2.2.2 Low-Fi prototype user testing

Second, the project required the potential end users' feedback in regards to the proposed application functionality and design which was based on research questions from Chapter 1.2. These questions specifically aimed to answer the second and third questions regarding application usability and systems adoption towards the case study company. Also, this approach was chosen to extract the most relevant information from the potential future users of the system. The goal of the system was already defined in Project Motivation in Chapter 1.1 and its research question from Chapter 1.2. Consequently, this vision was provided to the participants in the form of a Low-Fi prototype on top of which constructive criticism could be applied.

As a result, the interview flow consisted of two parts. At first, for a better understanding, the participants were introduced to the concept of the proposed solution. The participants were given a walk through of a preliminary design of the solution presented in the prototype. The Low-Fi was split into two scenarios that users needed to complete while using Think Aloud [15]. The first scenario was to create a Test Case; the second to Edit one of the existing Test Cases. The representations of these scenarios were done by

using InVision [16] prototyping application. Afterwards, a set of 13 questions was formulated and is available in Appendix G. They include general comments to the project idea and input regarding the Low-Fi prototype and its functionality.

2.3 Secondary research

The objective of this research was to find academic articles and books related to the problem formulation of the project and its scope. Therefore, secondary data research was conducted using the Aalborg University Digital Library [17] and Google Scholar [18].

For the purpose of literature gathering, a set of keywords were used in previously described tools. This list was based on the information in the research question and the project motivation. Such keywords included: Web technologies, Text context analysis, Neural networks (NN), Text vectorization, and Text semantic analysis.

2.4 Implementation independent model and Requirements

In the chosen process model, it was necessary for a software solution to have a set of requirements. For the purpose of the requirement development, different type diagrams were used to visualize the system's entities and flows. These diagrams established a graphical overview of the project's solution.

The requirements for the final solution were derived from the **primary and secondary** research which was initiated to produce the guidelines for the project's development decisions, specifically, in Chapter 5 and Chapter 6. These chapters describe the final solution of the project in regards to answering the research questions.

Furthermore, the MoSCoW prioritization scheme was selected for the project's use. This scheme has a four-level prioritization model. These priority classifications are: Must; Should; Could; and Won't. Each of these classification categories is focused on the project solution's requirements keeping in mind their differences and importance. [19]

- **Must have** - such requirements are fundamental for a project to be considered as success; therefore they must be included in the final prototype.
- **Should have** - such requirements will be improving the final prototype; however, they are not essential to the system.
- **Could have** - such requirements represent the full scale solution that could be implemented in the project's case study company - GN ReSound.
- **Won't have (for this time)** - such requirements will not be included in the prototype development. However, they should be addressed as a future improvement for the later development.

After all, requirements were specified, they were placed into corresponding table sections based on the MoSCoW scheme. Afterwards, the list was split according to the functional (FR) or non-functional (NFR) requirement type. In addition the requirement tables consisted of four fields that provided more details for the specific requirement. Such fields were:

1. **ID** (FR/NFR No.) is used to identify the sequence of requirement type.
2. **The "System area"** is showing the development field of where the requirement originated from (Database | Server-side | Client-side | Machine Learning).
3. **The "Source"** is pinpointing the chapters of the report from which requirement was extracted.
4. **The "Description"** is elaborating further on the requirement's application.

2.5 Testing

To finalize the developed solution, it was decided to execute different testing processes that would validate the performance of the solution. First of all, the performance of the Machine Learning algorithm was tested using a self-generated data set. The back-end of the TTTM system then underwent software testing to verify that its functionalities perform the expected way. Finally, User-Testing was planned to gather user feedback for the solution and verify it against the users' expectations.

2.5.1 Machine Learning Testing

It was performed to verify its accuracy in different use cases. To do this, a set of Test Steps, semantically similar to the ones stored in the database were created. The criteria of success was that the TTTM Machine Learning system must identify which database Test Steps values are similar to the user input values. This test passed successfully even when the ML responded with more than one similar Test Step, however, only if the response contained the anticipated answer. The test was failed only when the TTTM system would not be able to include the correct Test Step in the response. This test was performed twice to identify whether the ML data set update improves the test results. The outputs from the first iteration were then used to modify the ML model with additional data entries, then the Machine Learning testing was executed again using the same inputs.

2.5.2 Black box Software testing

It was executed to verify that functionalities of the back-end perform the way they are described in Chapter 5.3 and Chapter 6.3.2. Each test was designed to validate one or several API endpoint functionalities. Eight test scripts were developed in the server-side framework, and JSON objects were created as inputs to the API requests.

2.5.3 User-Testing

It was prepared to verify that the product prototype of this project would satisfy the needs of end-users, who as stated in Chapter 1.2 were employees of GN ReSound. The test was designed around different user scenarios, which were based on fulfilling MUST and SHOULD requirements of the TTTM system. The Test Cases included the steps to execute, the expected outcome and relevant requirement identification numbers.

Chapter 3

State of the Art

This chapter presents the summaries of the researched literature and technical documentation found on previously mentioned topics in 2.3. The chapter starts by explaining the domain knowledge of the test framework currently employed by GN ReSound. Afterwards, the overview of the possible technologies and existing solutions that are connected to the research question are introduced. Another part of the research summarizes the currently viable general Machine Learning algorithms and Neural Networks that can be applied to the research question. The chapter ends with an overview of the candidate technologies that can be used for the implementation of the project both for implementing Machine Learning algorithm and the Web application as defined by the project's research question in Chapter 1.2.

3.1 Project's development premise

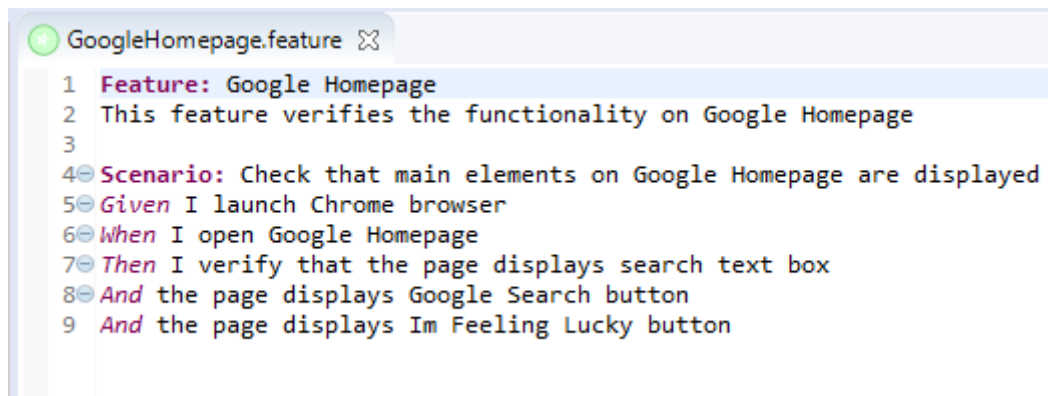
This chapter introduces the framework that the case study company - GN ReSound currently uses for their UI test cases. It gives an idea of the environment where the TTTM is to be implemented. This environment also defines some of the solution's requirements.

GN ReSound is currently using a development approach called BDD or Behaviour Driven Development. It can be viewed as a combination of practices which aims at reducing activity amount in software development that can be considered "wasteful" [20]. The goal of BDD is to optimize the communication between the development team members and the customers. In this communication, real-world examples are provided for better understanding of development goals. The concept of BDD can be split into two subcategories - Deliberate Discovery and Test Driven Development (TDD).

Dan North, the creator of BDD, once said - "Ignorance is the single greatest impediment to throughput" [21]. A big part of Deliberate Discovery is finding out how different project development stakeholders understand system behavior. Discussions and communication between the parties narrow the direction of the project, which reduces the potential rework in the development process. In this step, the involved participants exchange the function-

ality of the system using real-life examples. These examples show the system's behavior and can be later used for automated test creation.

Test Driven Development is the process when the automated test cases are created before the actual system implementation. TDD is compatible with different test types such as Unit tests or UI tests. However, the most important feature of TDD tests is that they are written in natural language so that all necessary parties can understand them. The currently most popular TDD/BDD test writing language is Gherkin, which is supported in both Java and .NET software development. An example of a Gherkin test case can be seen in Figure 3.1.

A screenshot of a code editor showing a Gherkin test case. The editor has a tab titled 'GoogleHomepage.feature' with a green circle icon and a close button. The code is as follows:

```
1 Feature: Google Homepage
2 This feature verifies the functionality on Google Homepage
3
4 Scenario: Check that main elements on Google Homepage are displayed
5 Given I launch Chrome browser
6 When I open Google Homepage
7 Then I verify that the page displays search text box
8 And the page displays Google Search button
9 And the page displays Im Feeling Lucky button
```

Figure 3.1: Gherkin test case example [22]

The usual test case is represented in a feature file that describes one feature of the system. In this context, a system's feature is essentially one single functionality. This feature is then verified using one or several test cases, also referred to as "scenarios", which consist of test steps and are presented in textual form. Additionally, test steps can have user entered variables separated with " ' ". Each test step must have a prefix assigned to it. BDD currently provides four different prefixes. "Given" specifies the pre-conditions of the test case, "When" defines an action, "Then" presents an expected outcome, "And" is used to extend any of the previous prefixes [20]. The code scripts for these test steps are later defined in Gherkin based test framework by the development team.

3.2 Existing solutions

As BDD Frameworks utilize information in natural language text format, it narrowed the research of existing solutions to services that provide their users with the analysis of textual input. Only the most relevant functionalities of their solutions to the research question are presented in the below summaries.

3.2.1 TextRazor

TextRazor [23] is a company that provides a relation extraction module which utilizes a set of defined sophisticated linguistic rules that ensures high accuracy in different textual analysis types. These different analyses are performed using statistical Machine Learning algorithms. The TextRazor API functionality examples include entailment and synonym extraction, text relation extraction and classification.

The synonym extraction is based on the word and its meaning in the specific sentence. The approach of detecting the semantic usage of the word decreases the amount of false-positive results for words with different contextual meaning.

Classification functionality enables textual input to be classified under specific silos like "political", "finance" or "sports". This same functionality can be used to add classification tags to explore more topics the analyzed text is related to.

Text relation extraction is a functionality that creates a model of the words in an analyzed text reflecting the different relations between each other. It is achieved using a combination of hierarchical spanning tree parsers. The partial result of the text relation extraction can be seen in the example in Figure 3.2. This example is using the following sentence as an input: "The bank announced in 2008 that Manchester City owner Sheikh Mansour had agreed to invest more than £3bn."

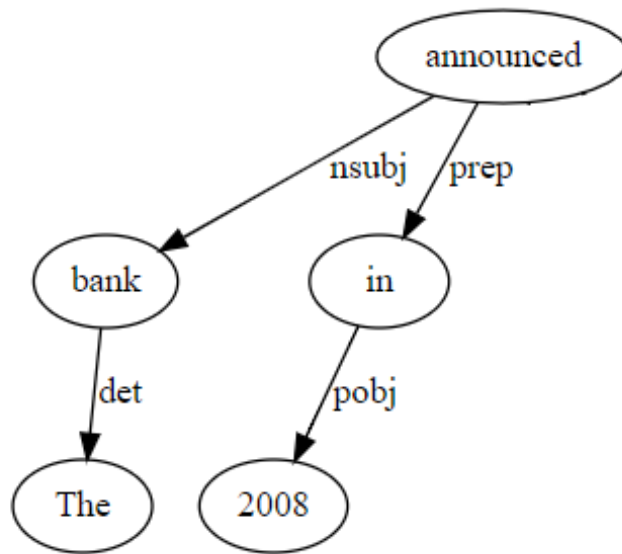


Figure 3.2: Relation tree of an analyzed text [23]. The full figure can be seen in Appendix A

3.2.2 Aylien

Aylien is a company that provides Machine Learning based Natural Language Processing (NLP) APIs and platforms [24]. Their API stack covers four different functionalities - sentiment analysis, categorization, extraction and processing. The most relevant APIs for the presented research question are Entity Extraction and Summarization endpoints.

Entity Extraction, together with another endpoint called Concept Extraction are utilizing linked data and Wikipedia API [25] endpoints to describe different entities in an analyzed text. These endpoints provide the URLs that can be used to retrieve the description of different entities. In addition, the URLs provide the specific entity's classifiers or types that explain its use in a given context, e.g., building, company, fruit. An example of retrieved information from the sentence "Apple was founded by Steve Jobs, Steve Wozniak and Ronald Wayne" can be seen in Figure 3.3.

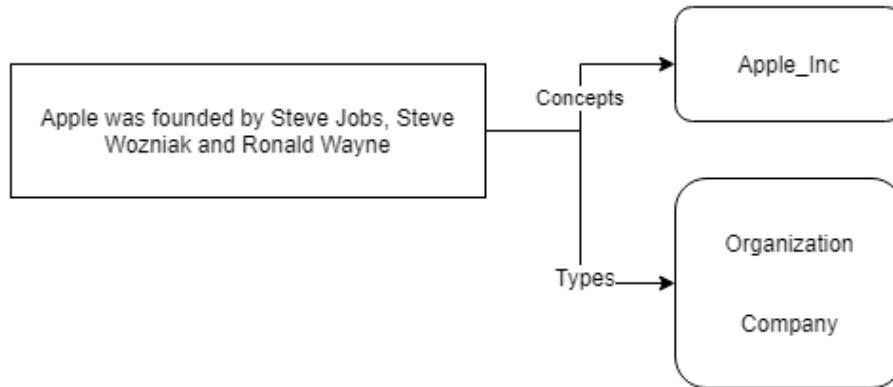


Figure 3.3: Example of Concept extraction API endpoint output of Aylien

The Summarization API endpoint specializes in summarizing a textual input into a few key sentences. The specific number can be defined by the user. The API can provide output for both textual input and URLs directing to a textual input.

3.2.3 Dandelion

Dandelion is another company that provides different RESTful API endpoints for textual input analysis [26]. Their API presents such functionalities as Entity extraction, text and content classification, Sentiment analysis, Concept extraction, Semantic similarity and Article extraction. Out of these, the most notable in relation to the presented research question in Chapter 1.2 are Entity and Concept extraction, and Semantic similarity analysis.

Entity and Concept extraction API endpoints have similar outputs, however, it can be said that Concept extraction provides them in greater detail. The Entity extraction discovers named entities in the text and tries to identify them in their respective context. This functionality is using Wikipedia API [25] to determine the meaning of each entity. An

example can be seen in Figure 3.4. It is made using a sentence "Apple releases new iOS version". The figure identifies the most important words and makes a Wikipedia search for each of them to identify their meaning.

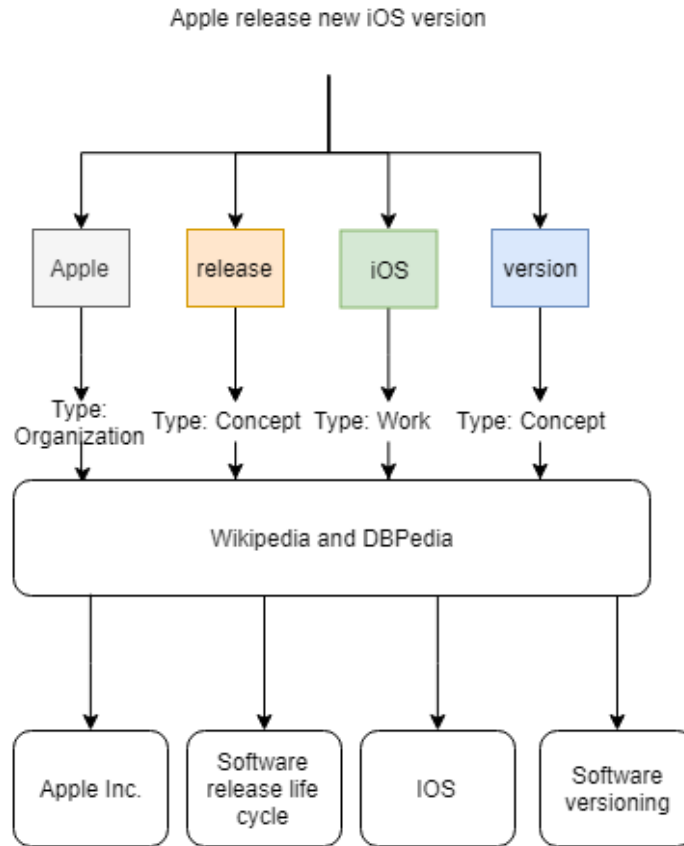


Figure 3.4: Example of Entity extraction API endpoint output of Dandelion [26]

Semantic similarity API endpoint provides the user with two similarity scores for two different texts. This API analyzes the semantic and syntactic similarity between two textual inputs presented in percentages.

3.3 Machine Learning Algorithms

The previously described existing solution show a trend of Machine Learning utilization in textual analysis. Therefore, it was decided to research deeper into Machine Learning capabilities in terms of textual analysis. This chapter presents an overall introduction of Machine Learning and its different types that are relevant to the research question of this thesis.

3.3.1 Machine Learning Overview

Machine Learning is a systematic way to make machines learn based on models, not rules. It is suitable for pattern finding, input based prediction generation and achieving goals based on data set's values and mathematical algorithms. It serves the purpose of automating complicated processes and making human life more comfortable. There are four main learning types according to which machine learning could be implemented. Each type prevails in different scenarios with different goals due to the diverse algorithms and learning models [27].

1. **Supervised learning** - The whole process is based on training the system with a data set that includes well-defined and known inputs and labeled Outputs. The trained system later would be used to predict the output values on the new data Inputs, which were not in the training set [28].
2. **Unsupervised learning** – This process is used when the data set only consists of inputs on the contrary to supervised learning. As a result, the system is looking for patterns or structure of the input to cluster the output [28].
3. **Semi-supervised learning** – This approach is a mixture of supervised / unsupervised learning. Its data set is populated with Inputs and some labeled outputs [28].
4. **Reinforced learning** – This kind of learning differs from the ones mentioned above. This type is initiated by the system's interactions with specific tasks, f.e navigation in the environment [28].

After exploratory research, the following ML algorithms were chosen as candidate technologies for additional research - LDA, LSA-SVD and HMM. All of these algorithms belong to Supervised learning type.

3.3.2 Latent Dirichlet Allocation (LDA)

The LDA [29] is based on probability models which are capable of performing document classification based on text segments. The document might be a simple string containing words or a huge article. The final generated model is represented by a probability matrix that maps words to topics.

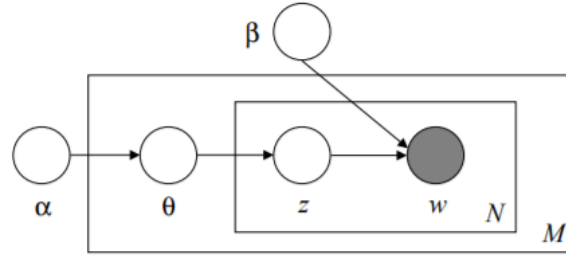


Figure 3.5: Graphical model representation of LDA [29]

Figure 3.5 presents how LDA model looks. The N block presents words and M block presents documents. The Theta parameter determines topic distribution in a document; while Zeta defines the topic per word in a document. In addition to that, Alfa and Beta are Hyperparameters of Dirichlet distribution [29]. Alfa defines document topic distribution, the higher it gets, the more topics the document contains. The Beta defines per topic word distribution, which implies that the bigger the value gets, the greater amount of words is distributed in a topic.

LDA could provide this project with the following benefits:

1. LDA provides feature extraction that assigns words to labeled topics, where each word can be linked to multiple topics.
2. It is a modular and extensible algorithm.
3. Great performance on previously unseen documents.

LDA has the following constraints:

1. It does not perform well with sequential inputs.
2. This algorithm does not perform well with documents that are short with words.
3. It does not determine a correlation between distributed topics.

3.3.3 Latent Semantic Analysis - Singular Value Decomposition (LSA-SVD)

LSA-SVD is a competent algorithm that can be applied in text summarizing. As defined in the research paper [30], written by Y. Gong & X. Liu, it can determine the similarity of the words and differentiate between sentence contexts. A quote from this research paper also states that *"The two synonyms doctor and physician generally appear in similar contexts that share many related words such as hospital, medicine, nurse, etc."* [30]. Therefore it can be concluded that similar words would be combined and mapped close to each other in a singular vector space.

LSA-SVD could provide this project with the following benefits:

1. This algorithm is capable of performing textual data reduction and semantic analysis.
2. This algorithm has a relatively small training time.
3. It is easy to implement and use.

LSA-SVD has the following constraints:

1. It does not perform well with sequential inputs.
2. This algorithm performs poorly with words that contain multiple meanings.

3.3.4 Hidden Markov Model (HMM)

HMM is a model that is capable of recognizing patterns, sequences and structures of the dataset's elements. Its models are based on probabilities and the training dataset through which predictions are generated [31]. For example, this model can perform text or sentence analysis based on input's structure and predict the next value in the sequence [32]. The prediction process cannot be observed as HMM utilizes hidden states.

HMM could provide this project with the following benefits:

1. It is easy to train a model because it does not require a big amount of training samples.
2. This algorithm does not require much computing power.
3. Has the ability to handle sequence based inputs.

HMM has the following constraints:

1. It gets sensitive when it deals with dimensionality reduction on the input data set.
2. A complex algorithm that has a number of different parameters such as hidden states and probabilistic weights that increases the training time

3.4 Neural networks

Another type of machine learning that can be applied to solve the research question of the project is the Deep Learning algorithm, also called Deep Neural Network (DNN) [33]. It is also known as a feature learning model that maps the input features to output. The most basic representation of a Neural Network can be seen in Figure 3.6, where there is an input layer, one hidden layer and an output layer. DNNs are Neural Networks (NN) that consist of multiple abstract hidden layers that each contains numerous neurons. These neurons are

mathematical computational units that in combination discover the relationship between the Inputs and Outputs. There is a multitude of problems that can be solved using Neural Networks, like image recognition and text prediction.

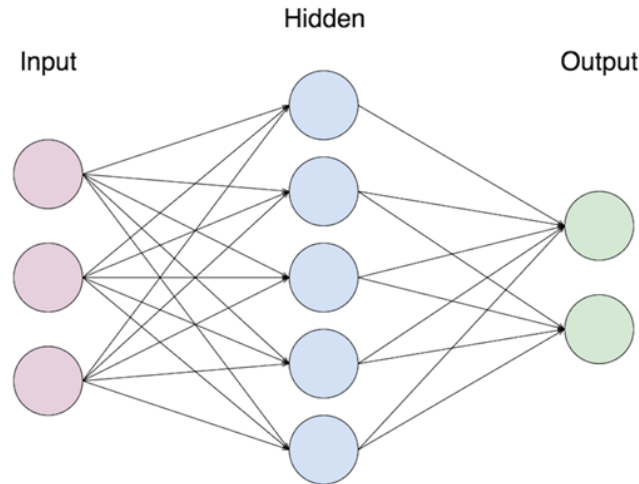


Figure 3.6: Example of a simple Neural Network[34]

Each node in NN also has an associated activation function which is used in combination with the weighted sum of the previous layer nodes. These activation functions include Sigmoid, Tahn, ReLU and Softmax [35]. **Sigmoid** - This activation function is used to determine probabilities. The output of the function is a value between a 0 and a 1. **Tahn** - This activation function is similar to Sigmoid, however, it is mostly used when the desired output can be negative. It results in a value in the scope between -1 and 1. **ReLU** - Rectified Linear Unit is an activation function that is currently highly used in Deep Neural Networks. It compares its input value with a 0 and outputs the highest of them. **Softmax** - This activation function is used in the output layer of the NN. It normalizes all of the output features so that the values are all between 0 and 1 and their sum equals to 1.

The activation functions are considered part of the Hyperparameters related to Neural Networks [36]. Other such parameters include the number of hidden layers, batch size which determines how much data the network will train itself each iteration, number of epochs which reflects how many iterations will the network be learning, dropout ratio which sets the probability with which the value will be passed to the next layer and the NN learning rate which determines the change of the model weights.

There exist a number of different Neural Architectures. However, there are two that perform textual analysis best which is a task directly connected to the proposed research question in Chapter 1.2. Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) have shown the capability of good textual analysis [37]. However, both pose challenges in regards to working with textual input.

3.4.1 CNN

Convolution Neural Networks are mostly known for their ability to analyze images. However, CNNs have shown promise in analyzing the semantics of textual input. One way to do it is to create sentence matrices similarly to image representation matrices [38]. One of the axes represents the amount of words in the sentence and the other the vector representation of the sentence. CNNs are usually used in tasks where it is necessary to identify the valence of the textual input. However, Convolutional Neural Network needs a specific size input for optimal performance that indicates challenges for it to work with different length sentence inputs [37]. It would require padding or resize of the input data to operate.

CNN could provide this project with the following benefits:

1. CNN performs well in classification problems and sentiment analysis.
2. It deals well with input data that is considerate noisy.
3. This algorithm handles well misspells and custom words.
4. CNN produces small sized models.

CNN has the following constraints:

1. It is computationally costly.
2. CNN requires good GPUs for fast training.
3. The network requires a lot of training data.
4. It is not optimal for working with sequential data.

3.4.2 RNN

RNN is usually used for sentence completion prediction, language translation and sentiment analysis [39]. In comparison to CNN, the RNN achieves the ability to deal with varying length inputs by having a connection between the input hidden units and outputs. It means that the hidden layers of this NN change after each word or letter of the input. Each word is processed at a specific timestamp so that the input can directly interact with the hidden state of the previous timestamp. RNNs are also able to provide an output for each timestamp, e.g., sentence completion prediction [40].

In Recurrent Neural Networks the later words in a text sequence are more dominant than the earlier ones which means that the RNN might not capture the key-words of the sentence correctly. This problem is also known as Vanishing Gradient Problem [41]. It persists in the earlier layers of the RNN, which means that RNN can forget what has happened in more extended sequences. Two concepts were introduced to solve this short-term memory problem - Long Short-Term Memory (LSTM) and Gated Recurrent Units

(GRU) [39]. Both of these concepts introduce different gates that have Tanh and Sigmoid functions [42] that decide which information is not important for the sentence and which can be forgotten. These two types are illustrated in Figure 3.7 below.

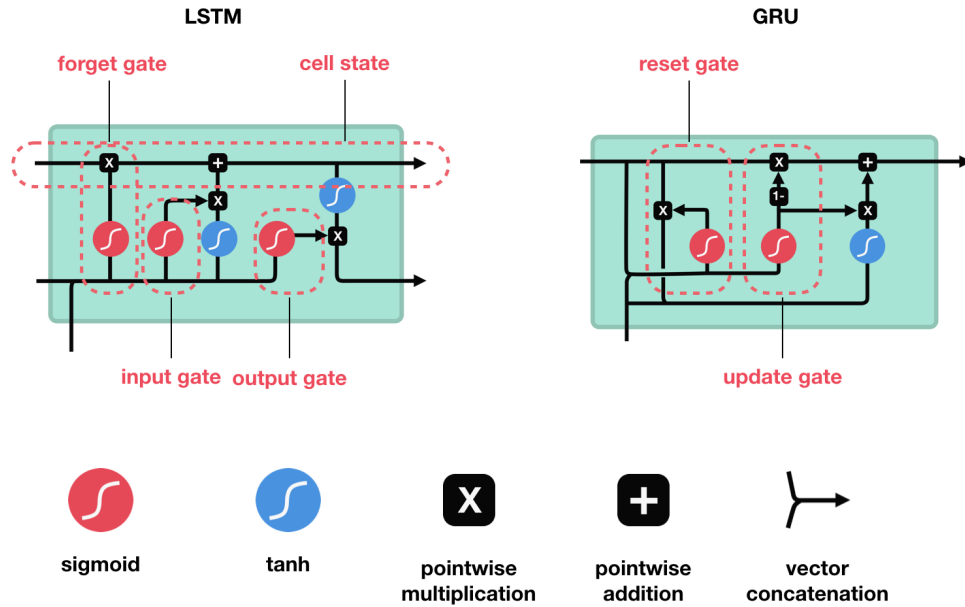


Figure 3.7: LSTM and GRU RNN types[43]

Both of these algorithms employ internal mechanisms called gates that regulate the flow of data [43]. In LSTM the Cell state is responsible for providing only the vital information for the RNN. The Forget gate uses a Sigmoid function that can squish the data between 0 and 1. Values close to 0 identify that the data must be forgotten. Input gate uses only essential data from the inputs using Tanh and Sigmoid functions. The output gate specifies what data should be carried to the next activation function. GRU, on the other hand, has only two gates. Update gate is responsible for forgetting unimportant information and adding new. Reset gate also serves to decide what old information from previous layers to forget.

RNN could provide this project with the following benefits:

1. This algorithm is able to analyze sequential data.
2. RNN provides a simple network structure.
3. It is considerate as a State-of-the-Art choice for Natural Language Processing problems. [39].

RNN has the following constraints:

1. Training RNN requires a lot of computation resources.
2. It consumes a large training time.
3. This algorithm requires a lot of memory for models.
4. The RNN model can be easily overfitted.

3.5 Machine learning tools

To prepare and add a Machine Learning (ML) algorithm to the system, it was required to create and test ML models. For this purpose, several existing solutions from the current business and academic practices are introduced to determine the most suitable tool for this project's purpose. These solutions are described in a form of summaries and the most relevant bullet list points for the development of the project's prototype.

3.5.1 TensorFlow

TensorFlow is a published, open-source, end-to-end platform for building and deployment of machine learning models [44]. The platform utilizes a combination of Python front-end and C++ back-end to help users build and deploy their machine learning models. In the time of the writing of this report, TensorFlow has been updated up to version 2.0. This version allows for the users to make and deploy their models to a wide variety of platforms such as Node.js server, Android or iOS device and IoT devices.

This version allows users to utilize different levels of abstraction. The newer and less experienced users can use Sequential API that allows constructing models from preprocessed building blocks. The Subclassing API, on the other hand, allows advanced users to approach their model development on a lower level. This API enables customization of layers, activation functions and training loops.

1. Capable of creating both, machine learning systems and neural network models.
2. Fast training performance.
3. Can provide medium and low level API.
4. Provides in-depth system customization.

3.5.2 PyTorch

PyTorch can be considered very similar to TensorFlow; however, unlike TensorFlow, it is built using Python back-end. In addition, PyTorch has been developed and used by Facebook. PyTorch capabilities include Hybrid front-end, Distributed Training and Dynamic

development [45]. The Distributed Training allows PyTorch projects to have asynchronous execution. Finally, Dynamic development can be seen as a design-as-you-go approach where the user can change the computational graphs during runtime. Even though it is made entirely using Python, PyTorch does provide a C++ front-end option. PyTorch is generally used in fast phased projects.

1. The preferred tool for academic research.
2. Gives access to pretrained models on top of which it is possible to expand.
3. Provides a low level API.
4. Gives access to creating custom expressions.

3.5.3 Microsoft Cognitive Toolkit

Microsoft Cognitive Toolkit, also known as CNTK, is a deep learning framework developed by Microsoft [46]. Similarly to TensorFlow, it utilizes graphical structures to describe dataflows. CNTK is explicitly focused on deep learning Neural Network development. This framework also employs APIs for defining networks, training and evaluation that are compatible with Python, C++, C# and Java.

1. Different length inputs, like text, do not need preprocessing.
2. Provides transparency of the layers of RNN.
3. Is designed to work with C# natively, which makes it fully compatible with other Microsoft based solutions.

3.5.4 Keras

Keras is a Python based tool, which provides a high-level Neural Network API [47], enabling fast prototyping. Its main features are considered to be user friendliness and modularity. To increase usability Keras API minimizes the tasks the user needs to perform and provides clear feedback upon error encounters. The modularity of Keras is achieved by deconstructing models to configurable modules. These modules can be any part of the Machine Learning model, such as neural layers and activation functions. Usually, these modules can be added to the model by a single code line. Keras API can also be used together such systems as TensorFlow and CNTK.

1. Optimized to work with limited data sets.
2. Can be used together with a variety of back-end systems
3. Provides a user friendly interface on a high level API
4. Available for rapid prototyping.

3.6 Web service frameworks

After performing research connected to the first sub-question of the research question related to Machine Learning, the relevant information for the second research sub-question was investigated. Specifically different web development stacks like MEAN, MERN and LAMP are presented as candidate technologies for developing the web application part of the TTTM solution. In addition, all discussed stacks are listed as trending technologies due to their popularity in the market by usage, seen in the survey of 2018 [48]. Below, each stack is identified and described together with the purpose in the industry.

MEAN stack: This stack is allowing developers to prototype the web applications entirely relying on JavaScript (JS) programming language. It consists of four entities - MongoDB, Express framework, AngularJS client-side framework and Node server-side framework [49] seen in Figure 3.8. Each entity of this stack is open source and has strong community support with many publicly available free-to-use library modules. In addition, this stack is allowing the development of end-to-end applications. Furthermore, it includes a non-relational database that is using JSON data transmit format [50]. Since the same programming language is used across MEAN stack entities, it provides projects with good compatibility between each stack entity, scalability, optimization for cloud environments [50]. Using MEAN stack enables the applications to utilize the Model-View-Controller (MVC) programming pattern. Also, this stack is widely used for creating Single Page Application (SPA) systems.

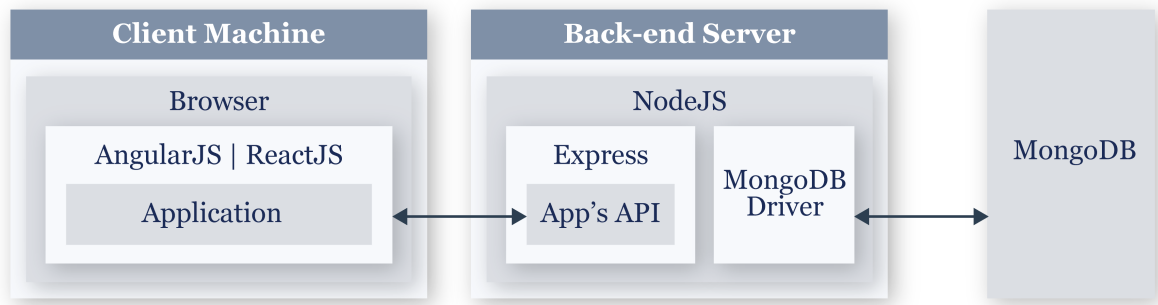


Figure 3.8: MEAN/MERN Stacks

MERN stack: This stack is nearly identical to the MEAN stack, as the majority of its entities remain the same [51] as seen in Figure 3.8. Besides, this stack is performing equally strong as MEAN and is keeping the same essence in the end. The only difference between them is that MERN stack's choice in client-side development is replaced from framework to a library. AngularJS is replaced with ReactJS due to its lightweight infrastructure and module reusability. Also, it carries the same advantages and goals to suit the SPAs.

LAMP stack: LAMP stack is differing from the previously mentioned stacks in the technological choice. In fact, it is one of the first open source stacks, which is widely used nowadays for web application development [52]. It consists of the following four entities - Linux, Apache, MySQL and PHP, see Figure 3.9 below. In industry, LAMP is widely used for making Content Management Systems (CMS) as it is powerful, cheap to maintain and most importantly, reliable [53].

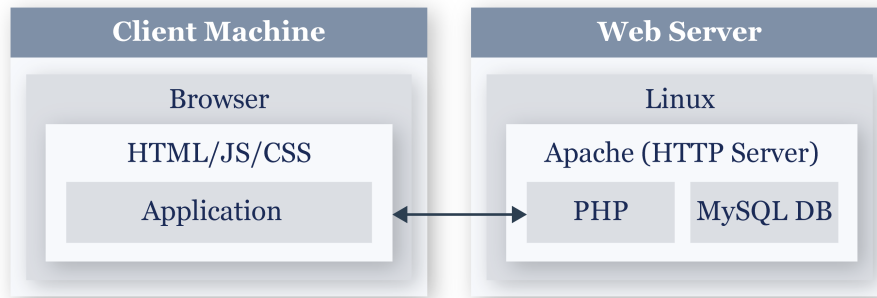


Figure 3.9: LAMP Stack

Chapter 4

Analysis

This project's intention was to connect two groups of people that participate in the QA process and to automate solution testing. Specifically, these groups are developers who write test scripts and Test Case documentation experts who define what functionalities application contains. Both of these groups are perceived as TTTM system's user base. After the initial research, the presented materials were analyzed to determine decisions for the design of TTTM's system. In the start, an overview model of the solution was constructed using the specifics of the project's motivation and expected outcome from Chapter 1.1 and Chapter 1.4. The TTTM system's overview model presented in Figure 4.1 shows both the Machine Learning and the Web Application entities in relation to research question sub-points in Chapter 1.2.

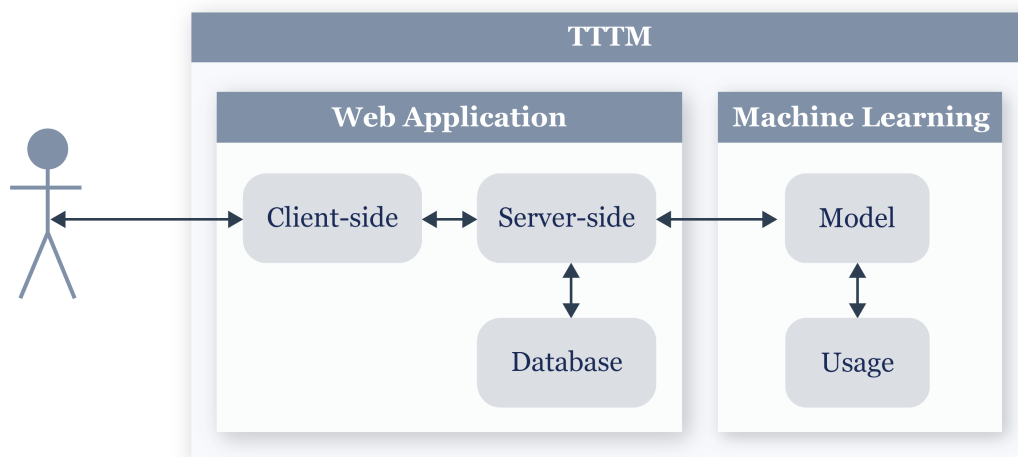


Figure 4.1: Solution concept diagram

The Web Application is divided into Client-side, Server-side, and the Database [54]. The Client-side serves as an interface for the end-users to create Test Cases. The Database is where the necessary user, Test Case, and Test Step information can be stored. Finally, the Server-side is the control unit that handles data exchange flows between Client-side, Database and also Machine Learning entity. The Machine learning part is divided into the actual Model that is used for the TTTM system's goals and the Usage - specifying the way how such model should be created and trained. Figure 4.1 shows an overall setup of the TTTM system. To fill in the details of each component, the subsequent chapters present an overview of how were the concurrent decisions for this diagram's entities made.

First, a background overview of automated Test Frameworks is presented to establish GN ReSound in the context of existing testing frameworks. Second, the comparison analysis between different web technologies is performed. This discussion results in the most relevant web technology stack for the TTTM prototype. Next, the most suitable Machine Learning algorithm is chosen out of the ones presented in Chapter 3.3 and Chapter 3.4. After the choice is made, the most suitable tool for developing the Machine Learning system is then chosen. Afterwards, the chapter continues with the analysis of primary research presented in Chapter 2.2. This analysis presents the summary of user involvement phases and features requested by the users. The chapter proceeds with the requirement specification that summarizes the system behavior using UML diagrams and a list of requirements.

4.1 Background of Analysis

There are many existing solutions, services, and frameworks in the market which aim to automate test environments. Such solutions usually are split based on the testing method, for example, BDD and TDD in GN ReSound company's case mentioned in Chapter 3.2. The most popular platforms for providing test automation based on the previously mentioned categories are Gauge (BDD/TDD) [55], Cypress (TDD) [56] and Serenity (BDD) [57]. Even though GN ReSound uses BDD, it is an in-house developed solution that can only be accessed via employees' credentials in a form of company specific username and a password. The framework is dynamically regulated and adjusted based on the company's needs, therefore, none of the above-mentioned solutions are capable of fulfilling what this project intends to improve in correlation to the GN ReSound test environment.

Because of this, it is intended to build TTTM on top of the currently equipped BDD framework. The core of this project is focused on text similarity, specifically on Test Steps, which are relatively short length strings that can include different type of variables. Existing solutions that can evaluate text similarity are described in Chapter 3.2. The upcoming chapters delve into the analysis of technological choices of TTTM system.

4.2 Web stack comparison

This chapter aims to analyze the Web Technology stacks presented in Chapter 3.6, to answer the second sub-question of the research question by selecting the most suiting candidate technology for the development of the Web Application seen in Figure 4.1. This chapter provides the overview tables of the Web Technology stacks, specifically - MEAN, MERN and LAMP. These web stacks were taken into consideration due to their demand on the market as the most trending technology stacks [48].

The overview tables are filled with key features relevant TTTM system's architecture, are used for outlining the capabilities of Server-side, Client-side and Database, and can be seen in Appendix B. Table 4.1, Table 4.2, and Table 4.3 show partial representation of these overviews. At the end of this chapter, the best suiting web stack for TTTMs prototype is selected. The differences between these stacks are based on the specific features offered by the technology.

General				
Features	Description	MEAN	MERN	LAMP
Open source	Free public access, progressive software quality, security patches, adaptability.	+	+	+
One language per stack	Increase the Stack's stability and scalability over the platform.	+	+	-

Table 4.1: General overview table

General: The three mentioned stacks are open source and compatible with the Model View Controller (MVC) [58] pattern. However, the LAMP stack lacks features that the other two stacks can provide to the developers, for example, running on the same programming language, excellent stack stability, scalability and easy debugging.

Server-side				
Features	Description	MEAN	MERN	LAMP
Asynchronous I/O & Non-blocking	Fast, handles as many requests as possible, by not queuing them.	+	+	-
CPU intensive tasks	Performs well with tasks that requires a lot of CPU.	-	-	+

Table 4.2: Server-side overview table

Server-side: The server-side technologies of the stacks differ in terms of performance and architecture goals. The main similarities these stacks share are JSON compatibility and high performance of the application [59]. While the LAMP stack is more suitable for CPU intensive tasks such as large mathematical computations [59], MEAN and MERN stacks are more dynamic and scalable solutions in terms of request and response exchange performance [59].

Client-side				
Features	Description	MEAN	MERN	LAMP
Dynamic SPAs	Fast and efficient way of building web apps that don't reload on requests. Improves UX of the app.	+	+	-
Modern client-side model	Uses libraries and frameworks; not plain languages in files as: CSS / HTML / JS.	+	+	-

Table 4.3: Client-side overview table

Client-side: The LAMP stack cannot be compared in this category due to its design which is excluding the client-side framework. Although such frameworks could be adopted, it was not the point of this comparison going beyond the stack's structure. On the other hand, both MEAN and MERN stacks are meant to be used for creating Dynamic SPAs, which are widely used in the industry and are backed by such corporations as Google [60] and Facebook [61].

Databases: There were two types of databases used in the analyzed stacks. The LAMP with relational type, structured via tables and rows while MEAN and MERN databases belong to the document type, that was JSON oriented.

Discussion: After summarizing all web technology stacks visually by using tables, a conclusion was formed that each stack performance differed based on the purpose of the application.

Based on the overview tables, and backed by analyzed literature [59] it can be concluded that LAMP stack is outperformed by modern stacks like MERN and MEAN, due to their overall performance, usage of the same programming language, fast response times, scalability and modern approach for building web applications. MEAN and MERN both utilize similar structures apart from the Client-side technology where MEAN uses a Framework and MERN uses a View-library.

The project aimed to create a web application for test automation purposes. It was clear that such features as the system's scalability, performance, and integrity were required

from the stack to be selected, due to the Enterprise environment where TTTM solution would be equipped, as indicated in Chapter 4.1 and Chapter 1.1.

Therefore only the two similar stacks were identified as fitting candidates for the development of the prototype, MEAN and MERN. Since both of these stacks were different in client-side technologies and developed with intentions to provide SPAs by default, it was necessary to select stack that provided more benefits for the TTTM system. At the end of comparison, the MERN was selected for prototyping of the TTTM Web Application, as it had the following benefits – it is a lightweight library that does not include unneeded features to the needs of this project, provides better user experience and it can be adopted on top of existing web applications as a module or component.

4.3 Machine Learning vs Neural Networks

The purpose of this chapter is to identify which Machine Learning algorithm out of the presented ones in Chapter 3.3 and Chapter 3.4 is most suitable for solving the first sub-question of the research question presented in Chapter 1.2. This algorithm will be used to develop a Machine Learning model of the TTTM system as seen in Figure 4.1. As described in Chapter 4.1, the machine learning model should be able to perform a semantic text similarity comparison. The proposed algorithms are reviewed to determine the most applicable one to this project. This is accomplished by conceptualizing and analyzing the performance of the TTTM system with these candidates in use.

An overview Table 4.4 of the compared algorithms is presented below. The table includes six algorithms out of which half belongs to the Neural Networks (NNs). Moreover, it displays six features that are perceived to be beneficial towards research questions and final algorithm choice. In addition, each feature is rated on a scale from 1 to 3, where 1 is defined as a poor; 2 as moderate; and 3 as desirable performance in the context of the chosen algorithms.

1. **Accuracy rate** - this feature is providing an approximate level of the accuracy of the algorithm's predictions. The algorithm's accuracy rate is a substantial performance measure.
2. **Implementation simplicity / Computation resource optimization / Low training time** - these features are relevant due to agile development methodology, debugging and possible limitations in the prototyping.
3. **Compatibility with small datasets** - this feature is important because the dataset provided by GN ReSound and Expert users of the company is relatively small. In order to use the collected data as efficient as possible, it is necessary for the algorithm to be able efficiently utilize the small datasets.
4. **Ability to analyze sequential data** - this feature is required due to the nature of Test Steps and BDD presented in Chapter 3.1 and Chapter 4.1.

Features	LDA	LSA-SVD	HMM	CNN	LSTM Char	LSTM Sent.
Accuracy rate	2	2	3	2	3	3
Implementation simplicity	3	3	3	1	3	3
Computation resource optimization	2	2	2	1	2	2
Low training time	2	3	2	2	2	2
Compatibility with small datasets	3	3	2	3	3	3
Ability to analyze sequential data	1	1	2	1	2	3

1 : poor
 2 : moderate
 3 : good

Table 4.4: Machine Learning Features

The structure of the subsequent sections is based on the algorithm score for each feature in the table above. It is separated into Simple Machine Learning algorithms and Neural Networks. At the end of each algorithm's analysis, a conclusion is drawn to determine whether the current algorithm outperforms the previous or not.

4.3.1 Analysis of Machine Learning algorithms

This chapter elaborates on which of the Simple Machine Learning algorithms presented in Chapter 3.3 is best suited for the TTTM solution based on the summary Table 4.4.

Latent Dirichlet Allocation (LDA)

This algorithm was presented in Chapter 3.3.2, where general working principles, benefits, and constraints of it were defined in regards to this project. The accuracy that LDA can achieve ranges from 66% to 73% [62], [63], which is considered average in regards to this project domain that aims to identify most similar text fragments. In LDA most essential criteria for the model's accuracy is the size of the data set. As stated by D.M. Blei et al. [63] it is well compatible with data sets that are of various sizes. It is specifically indicated that 5000 documents are enough to maintain a 70% accuracy. This particular feature is relevant for this project due to the support of relatively small data sets provided by GN ReSound for this project. However, the fact remains, the greater the data set size, the greater is the chance of increasing the accuracy if the model is not being overfitted. In fact, since this algorithm is based on topic modeling, it is essential to provide it with many samples for each topic. It can be considered as a disadvantage for the project domain since the testing

topics of the Test Steps are custom in many cases and differ on each testing environment. The implementation of this algorithm is not complicated, due to the already published and well-tested representations of LDA written in Python [64]. This helps achieving the agile development, which is required by process model defined in Chapter 2.1. However, it is hard to predict how the algorithm would behave without proper prototyping phase, which includes tweaking and customizing tuning values. The training time of an LDA model varies, the larger the data set, the higher the training complexity. As an example, if the data set is as large as one million documents with one thousand words each, it can reach up to a few days of the model training time [65]. The main disadvantage of using LDA in this project is that it is based on a "bag of words" concept [63]. By using this concept, LDA does not follow the semantic meaning of the words nor has the ability to analyze sequential data. In addition, as stated in Chapter 3.3.2, it also performs poorly with a data set consisting of short input values.

Latent Semantic Analysis - Singular Value Decomposition (LSA-SVD)

As stated in a short review in chapter 3.3.3, LSA-SVD algorithm is widely used for text summarization purposes and is capable of mapping similar synonyms in vector space [30]. This approach is similar to the above-discussed algorithm - LDA as it is also using a "bag of words" concept for document analysis, which provides LSA-SVD with similar drawbacks to the ones stated above in the LDA chapter. There is a considerable amount of research done that is related to this project's domain. The most relevant papers with similar approaches to this project focused on the semantic similarity of textual documents [66], [67]. The average accuracy of LSA-SVD is similar to LDA. It ranges between 73% to 74% tested in the same use case as in LDA [62]. Generally, LSA-SVD is performing better than LDA due to the SVD part of the algorithm, which handles the dimensionality reduction on sparse data sets. One of the main problems of LSA-SVD is the scalability in production - as the algorithm needs to rerun the whole data set after the addition of a new document which would exponentially increase the server response time of the TTTM solution. However, it performs well with small to medium-sized data sets [68]. Also, it is easy to implement LSA-SVD due to the vast variety of existing libraries and research in that field. Since this algorithm is slightly better than LDA in terms of the few features defined above and visualized in Table 4.4, it can be concluded that LSA-SVD would outperform LDA in the context of the purpose of this project.

Hidden Markov Model (HMM)

The HMM algorithm is different from the previously analyzed algorithms, as it is capable of pattern, language or text recognition [69], [70] due to its use of probabilities that are spread over a series of observations [71]. The analyzed features of HMM are similar to the above-discussed algorithm features. The training time depends on the data set's size; generally, it is quite fast, even on older hardware [72]. Also, its implementation is as easy to achieve as LDA or LSA due to it having many libraries to choose from with the ability of additional customization [73] that would allow to adjust it to the needs of the project

domain. The downside of HMM is that it lacks literature for this project topic, based on the conducted secondary research. Therefore, the assumption was reached that HMM is not widely used for implementing text similarity tasks. Instead, based on observed literature, HMM is extensively applicable in speech [74], pattern [75] or gesture [76] recognition areas. Nevertheless, the inspiration on how it could be applied in the context of this project was drawn from multiple research papers focused on paired model creation and word similarity comparison [77], [78].

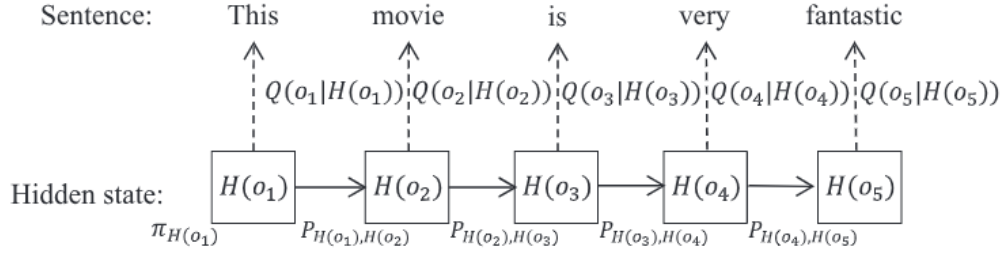


Figure 4.2: A text-based HMM [78]

Figure 4.2 provides a workflow of how the algorithm deals with sequential data. It is split into two states, the Input, and the Hidden state. The O represents words and Q - probabilities. The accuracy of such model text similarity is reaching up to 85% [78], which is a good score on this project's topic, as sentence prediction similarity would be appropriate to be suggested by TTTM system and trusted by the user. Even though the accuracy rate of HMM depends on the training data set, similarly to LSA-SVD and LDA, it requires more training data to achieve reliable accuracy score. Additionally, HMM is optimized to deal with sequential data, which is an important feature due to the project's need of deriving textual input semantics presented in Chapter 4.1. However, the hidden state of the algorithm minimizes the overall user understanding of the processes within the algorithm. Although HMM has most of the parameters similar to LSA-SVD, it can be concluded that HMM outperforms the previously analyzed algorithms due to the ability to deal with sequential data and higher average accuracy.

4.3.2 Neural Network analysis

As described in Chapter 3.4, Deep Neural Networks are a viable approach for solving the project's research question. There is a multitude of different CNN and RNN solutions that can provide results for the text similarity, nonetheless, three promising algorithms were found. These three algorithms were categorized as Multi-perspective CNN [79], Word-based LSTM network [80] and Sentence based LSTM network [81], introduced and discussed in regards to the project's goal mentioned in Chapter 4.1.

Multi-perspective CNN

The analyzed CNN algorithm consisted of two flows - feature extraction and feature analysis. At the start, the entered textual input is analyzed to extract features at different levels of granularity of the sentence. This approach utilizes the strengths of CNN mentioned in Chapter 3.4.1 by categorizing the textual input into the features. These feature vectors are then compared to each other using several similarity metrics [81]. The granularity is achieved by perceiving the input sentences as a $V \times W$ matrix where V represents the number of features of the CNN and W represents each word of the sentence. These matrices are then analyzed using two different methods - per-word and per-feature. The first method uses the word features in the form of a vector, where the second compares words by each of their feature value independently, as seen in Figure 4.3. In this figure, each circle is a value that represents how much of each feature V is present in each word W of the analyzed sentence.

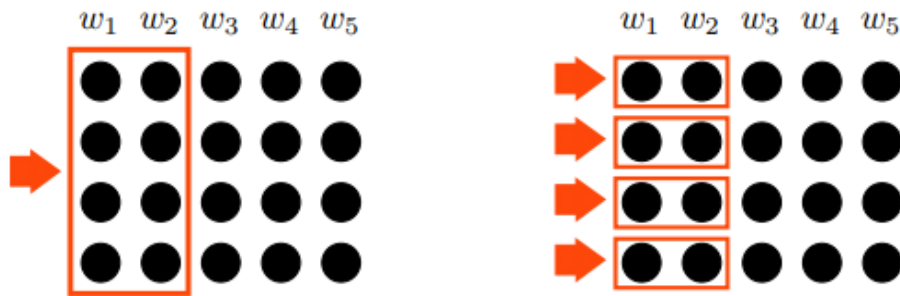


Figure 4.3: A representation of Multi-perspective CNN [82]

Even though, it is a customized Neural Network and this specific algorithm is designed for small data sets, contrary to usual CNN constrain from Chapter 3.4.1, it had only achieved accuracy of up to 79 % depending on the data set, which lower than in HMM. As a Deep Learning algorithm, it is resourceful in its training process. The designers of this specific algorithm also specify that it is fairly complex due to many information flows and a substantial amount of functional engineering constructed around the CNN model. The major disadvantage of this algorithm is that it is based on CNN that cannot grasp the connection of inputs in a sequential manner, as described in Chapter 3.4.1. Therefore, based on these arguments and Table 4.4, it can be said that this CNN algorithm cannot outperform the previously analyzed HMM algorithm.

Char based LSTM network

The Char based LSTM network was based on RNN and focused on phrase similarity based on normalized custom word phrases [79]. The goal of the researchers was to create a system that would be able to differentiate between different job titles. The system would be able to map user input job titles to a static set of predefined jobs.

The proposed network architecture in the research is a Siamese neural network and consists of four layers of Bidirectional LSTM nodes. A Siamese neural network is a network that consists of two sub-networks with same weights and joined output [83]. Each of the sub-networks extracts features from different input which solves the need of comparing two texts presented in Chapter 4.1. The joined output of Siamese network is the distance between the two feature vectors. Bidirectional LSTM is an extension of Bidirectional Recurrent Neural Network (BRNN) [84]. BRNN is an RNN that analyzes each value using its neighbor values. To put simply each input in BRNN not only affects the state of RNN for the future inputs but also for the previous ones, as seen in Figure 4.4. In this figure, X represent each word of the sentence while both A' and A represent the hidden layers of the network that are transferring hidden states of the network in both directions. Both of these hidden layer nodes are used to determine the outputs of the Neural Network Y for each word input.

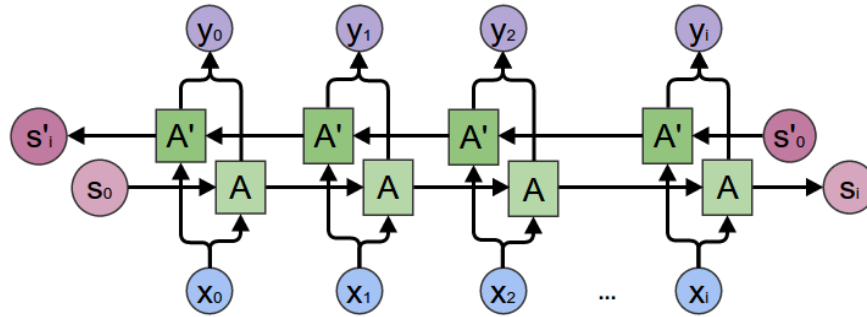


Figure 4.4: A representation of BRNN [82]

Figure 4.5 portrays how two different words are processed letter-by-letter through multiple LSTM hidden layers. After going through the BRNN layers, each result is altered to fit a fixed number of dimensions of the output and sent to a feedforward layer. This layer then sends the output to an energy function that determines the cosine of the two outputs of the Siamese Network.

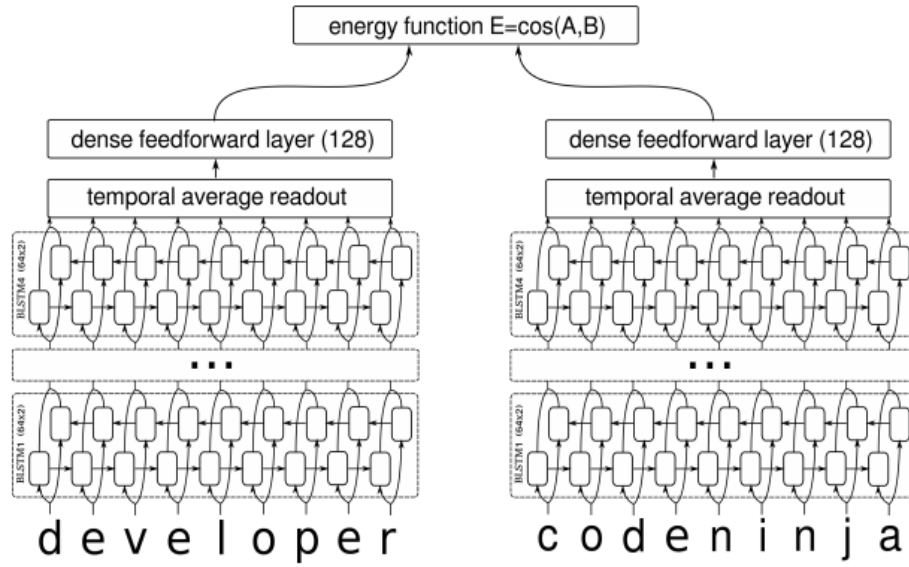


Figure 4.5: A representation of Char based Siamese LSTM for text analysis [79]

The research paper mentions that this algorithm deals well with finding similarities when the input has synonym words, extra words, and typos in regard to the predefined set. This feature can be utilized as the project contains a predetermined set of outputs that the user input needs to be verified against in the form of Test Steps. Moreover, this algorithm deals well with previously undefined word combinations, which allow the TTTM system to compute the similarity of Test Steps with custom word usage. As this algorithm is based on the wording of the inputs, the negative samples in the training data set should present examples that are most similar by syntax not similar by semantics. That will decrease the false-positive results from the Neural Network, e.g., "I set the volume to 9" will be similar to "I verify that the volume is 9". Apart from the purpose of the algorithm, it also provides an accuracy ranging between 82% and 100% depending on the use case as claimed by the authors of the research paper [79]. As seen in Figure 4.5, this algorithm has a simple implementation consisting of two bidirectional LSTM networks without extensive customization and a cosine function for similarity determination. This algorithm is also designed to work with small data sets. Together with the simplicity of the network, it identifies that the training time and resources are moderate for a Neural Network, which is also one of the general RNN constraints presented in Chapter 3.4.2. Fortunately, the design of the solution is not expected to be retrained dynamically. In comparison to HMM, this RNN algorithm is similar in performance. However, it can be argued that it was designed specifically to work with small data sets which is one point that HMM lacks. It can be concluded that Char based LSTM network can outperform HMM if the training data set is limited.

Sentence based LSTM network

Another LSTM that could solve the first sub-point of the research question in Chapter 1.2 focuses on deriving the similarity between different sentences [80]. This Neural Network is also a Siamese LSTM network and also uses a pair of strings and their similarity label as training data.

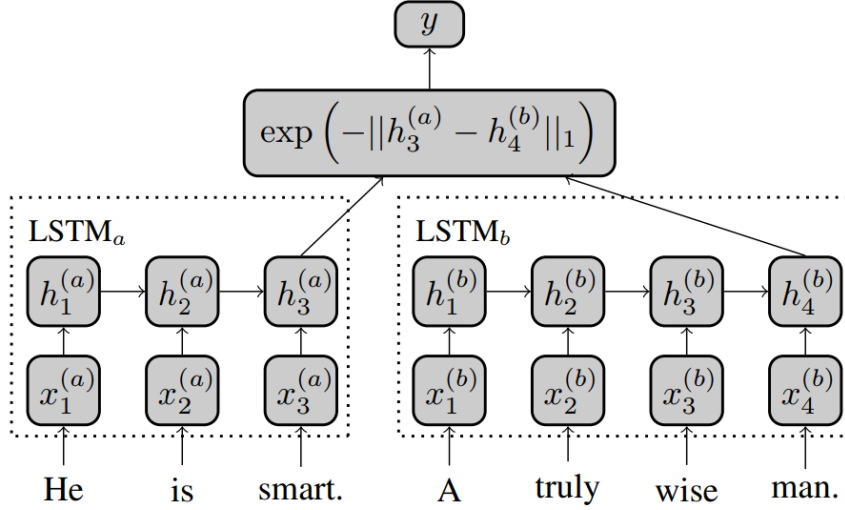


Figure 4.6: A representation of Sentence based Siamese LSTM for text analysis [80]

Figure 4.6 shows the structure of the proposed network. Each word of the sentence is vectorized in a number of the steps before entering the LSTM nodes. The word vectorization is done using the 300-dimensional *word2vec* embeddings created by Google [85]. The *word2vec* tool creates a vocabulary from the training data set. This vocabulary is used to determine the vector representation of new word inputs entered into this tool. There are multiple acknowledged ways to populate such vocabulary, for example, Wikipedia data sets [85]. After processing the determined word vectors in both LSTMs, the results are forwarded to an exponential function (Manhattan distance) [86] that results in a value from 0 to 1 which reflects the similarity of both sentence inputs.

This network can be described as simpler than the Char based LSTM network as it is not bidirectional. It is more oriented towards sentences; however, it lacks coverage of custom words and phrases as the *word2vec* vocabulary is based on a pre-trained data set. On the other hand in relation to the purpose of the ML algorithm in this project, this algorithm deals with natural language synonyms better not only based on the basic RNN design described in Chapter 3.4.2 but also due to the existing data set used for this algorithm. It can be argued that TTTM users will not use synonyms to the custom elements in the application, instead, they would use different sentence structure and style. Even though, the claimed accuracy of this algorithm achieves 84% in comparison to Char based LSTM network, it is on par with its training time and computation resource optimization.

Moreover, Sentence based LSTM network is also compatible with using small size data sets, which made it hard to determine whether the Sentence based LSTM outperforms the Char based LSTM.

LSTM algorithm comparison

Both of the LSTM networks showed high potential for solving the problem presented in Chapter 1.2. Both of these seem to outperform other presented algorithms. However, it is was not possible to determine the effectiveness of these LSTMs in the context of the TTTM system relying only on research papers and available theory, based on "No Free Lunch" theorem [87]. It states that *"there is no one model that works best for every problem"*. To determine which of the two presented approaches is better suited for achieving Test Step similarity analysis mentioned in Chapter 4.1, model testing was performed. For the purpose of comparison, instances of both Neural Networks were created and trained using the 16 Test Steps provided by GN ReSound found in Appendix D together with a text similarity data set from Wikipedia [88].

After the training period, the Sentence based LSTM network [80] reached an accuracy of ~ 0.92 and Loss rate of ~ 0.035 after 1.2 million steps that are equivalent to approximately 220 epochs seen in Figure 4.7.

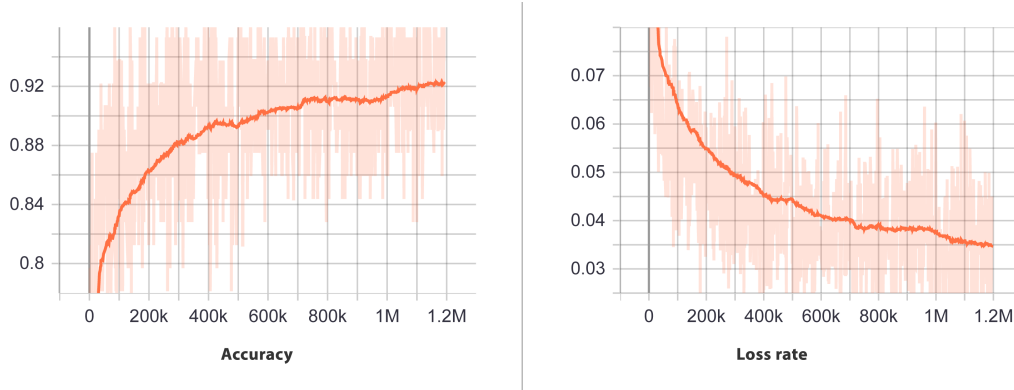


Figure 4.7: Sentence based LSTM - accuracy loss & rate extracted from Tensorboard [89]

On the other hand, the Char based LSTM network [79] at the same amount of epochs reached ~ 0.84 accuracy and ~ 0.06 loss rate, as seen in Figure 4.8.

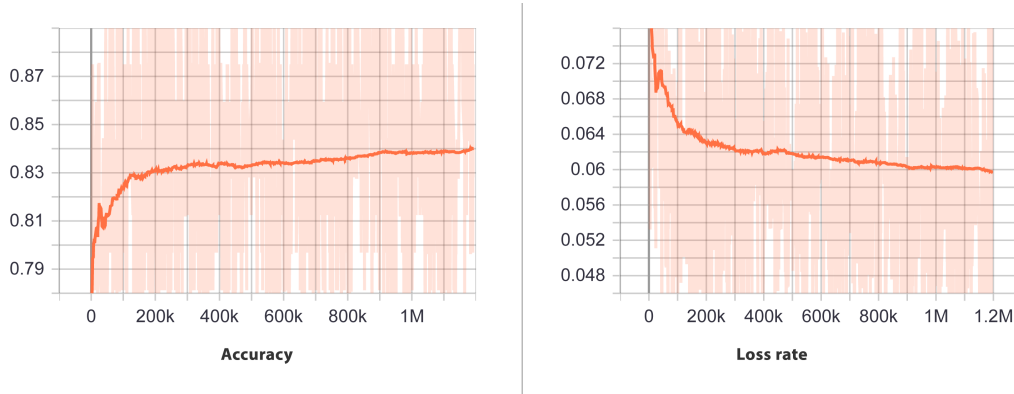


Figure 4.8: Char based LSTM - accuracy & loss rate extracted from Tensorboard [89]

Even after extending the training of this model to approximately 366 epochs (2 million steps) the accuracy reached a maximum value of ~ 0.846 and the loss rate the value of ~ 0.058 . After these values were reached, the model started overfitting, which is presented in Appendix F. Moreover, several model accuracy tests with contradicting input pairs were performed, which identified that the Char based LSTM network does not operate well with sentences consisting of words with different importance. In an example of two sentences - "Demo mode is present" and "Demo mode is not present" the Char based LSTM network recognized them as at least 80% similar, whereas the Sentence based LSTM saw these sentences as the opposite.

In conclusion, the Sentence based Siamese LSTM seems to be the better option out of two not only by the accuracy and loss rate values but also by the focus of the network. The Char based LSTM network was designed for deducing semantic similarity of phrases, specifically custom job titles. Even though GN ReSound testers use custom words representing different UI elements in the test cases, they use these words in the same word form, e.g., "All-Around" program is always called like that, examples like "Everything-Surrounding" are not used. That is why the Sentence based LSTM network that uses *word2vec* tool together with Wikipedia data set provides a better semantic understanding of Test Steps.

4.4 Tools for Machine Learning

To implement the selected algorithm, it was required to choose the most suitable tool which would serve the purpose of constructing and training the LSTM model as established in Figure 4.1. The decision between the previously presented tools in Chapter 3.5 namely TensorFlow, PyTorch, Microsoft Cognitive Toolkit and Keras, is based on the chosen algorithm, the data set in the possession and the overall design of the system.

One of the most popular and used tools is **TensorFlow**. It has a high variety of functionalities, including in-depth Neural Network layer and activation function customization

mentioned in Chapter 3.5.1. It provides mostly low-level API with many functionalities for experienced users. Even though TensorFlow provides fast and efficient training process, it is not considered as a necessary feature for this project; it is not expected to train the models for the solution in a continuous manner. The in-depth customization is also not required in the scope of this project as the Machine Learning algorithm is chosen from existing algorithms and customized only to the extent to be more familiar with Test Case structure of GN ReSound.

PyTorch is presented as the most used tool for academic purposes and also provides the ability to update existing pre-trained models, as summarized in Chapter 3.5.2. However, models in PyTorch similarly to TensorFlow are represented in a detailed and complicated manner as both provide low-level API. The selected ML algorithm provides a clear definition of hidden layers [80] which in return makes such a feature redundant for the implementation of the prototype.

Microsoft Cognitive Toolkit is a less known tool developed by Microsoft for Deep Learning model creation and training. In connection to this project, it has several substantial advantages such as no pre-processing needed for different input usage, e.g., different sentence lengths. On top of that, it provides additional clarity in the creation of RNNs seen in Chapter 3.5.3. The main disadvantages for the usage of this tool in the project are that it currently lacks documentation, apart from official Microsoft article, and it requires extensive knowledge of Neural Network structure on the lower and more detailed level. However, in the full-scale, solution this would be the perfect candidate for the TTTM solution as GN ReSound is mainly using Microsoft tools and services (Team Foundation Server or TFS [90] for organizational and documentation purposes and C# on Visual Studio as the main development language).

In the end, for the final prototype of this project, it was chosen to use **Keras** as the tool for Neural Network implementation. The main argument for this decision is the limited data set that was acquired in this project, which consisted of Wikipedia data set available online with the addition of GN ReSound Test Step formulation relation. From Chapter 3.5.4, it can be observed that **Keras** also does provide a more straightforward representation of the NNs with its high-level API. This enables fast and easy prototyping of the models, which goes well with the agile methodology of this project introduced in Chapter 2.1. Even though **Keras** does not enable the users to create custom hidden layers and lacks pre-trained models to build on top, such features are not necessities in the context of the TTTM system. This is determined based on the low complexity of the Machine Learning model required to solve tasks presented in Chapter 4.1.

4.5 Preparation of the data set for Machine learning

For the specified LSTM model, a data set was needed that would train the model to understand the semantics of different Test Steps. As the provided data set from GN ReSound consisted of only 16 Test Steps, it was decided to improve it with additional inputs

collected from users as described in Chapter 2.2.1. The involved expert users were GN ReSound employees from System Verification department as the final solution is designed with it as a case study specified in one of the sub-questions of the project’s research question in Chapter 1.2. These are the people who test all of GN ReSound’s products before deploying them to the market. These users tried to reach previously specified test goals using Smart 3D [14] mobile application that had its Demo mode settings on. Users were introduced with the test goals after which they performed the necessary Test Steps on a provided device while verifying their actions mutually. The audio files with the user input can be provided on demand.

Figure 4.9 shows the flow of one of the performed test goal completion. It shows the application states and the steps between them in a test goal "Change individual HI volume and mute the program". The steps in the box were created by the users, and the text above them represents the Test Steps from Appendix D.



Figure 4.9: User goal achieving process

First of all, the user needed to split the volume bars of the program. Next, they changed the value of one of the sliders based on their preference. Afterward, the user muted both volume sliders. Other user results can be found in Appendix C done in the same format.

The results of this phase provided additional 40 entries for the data set of the ML system. These steps can be seen in Appendix E.

4.6 Low-Fi prototype user testing

To add user insight to the TTTM system and to gather requirements for the Client-side of the project, a Low-Fi prototype was created. It was based on the first designs of the solution and was created and shown to users in order to understand their vision of the solution in the form of feature summaries. The main functionality of the prototype was to show the users different Test Step suggestions connected to their textual input as seen in Figure 4.10. The whole Low-Fi prototype can be found in Appendix H with its UI description or alternatively the Low-Fi prototype demo can be played on InVision webpage [91].

Validate Test Steps

Please define your test cases in the dropdown list below, then assign the values of its variables.

@ Change volume to 6 | ▾

COMPILERSAVE

Validate Test Steps

Please define your test cases in the dropdown list below, then assign the values of its variables.

@ Change volume to 6 | ▾

@ Volume of HIs to (.*)
@ Volume of Streamer to (.*)

COMPILERSAVE

Figure 4.10: Test Step validation process in Low-Fi prototype

Its testing was performed in GN ReSound with three of employees who are closely involved with their Automation Test Framework - a Senior Test Automation Engineer, a Software Test Developer, and a Senior QA Lead. The responsibilities of the user involvement participant roles are described in Table 4.5 together with the indication of which user contributed to which user needs.

Role name	CNN	RNN2
Senior Test Automation Engineer	The Senior Test Automation Engineer is the lead developer of the Test Automation Framework and is the person most familiar with the project's premised described in Chapter 3.1. He develops all of the code implementation of the Test Automation Framework.	#1, #2, #3, #4, #5, #6, #7, #8, #9, #10, #11, #12
Software Test Developer	The primary tasks of the Software Test Developer in relation to the Test Automation Framework is to create new test cases and define new test steps. His tasks are very closely related to the expected functionalities of TTTM.	#1, #4, #11, #12, #13, #14, #15, #16, #17
Senior QA Lead	The Senior QA Lead's tasks delve into creating and documenting Test Cases based on product requirements. He is part of the user group who do not have the knowledge of the implementation of the Test Automation Framework.	#1, #2, #9, #10, #11, #12, #13, #17

Table 4.5: GN ReSound expert user role description

All of the user inputs were summarized and added to Appendix I. These summaries pinpoint different user needs that are combined into a table of User Features. This list contains the most significant comments provided by the participants of the Low-Fi test. This table also presents an analysis of each feature, which is performed to establish if they are necessary for the TTTM requirement list. The features also reference the source test participants. An example of a singular feature is presented below in Table 4.6, and the full list of the 17 User Features is available in Appendix K.

Description	#12 - The system should be able to authenticate different users to provide users with the functionality of saving their Test Case drafts.
Sources:	Senior Test Automation Engineer, Software Test Developer, Senior QA Lead
Analysis:	The participants verified that such functionalities as authentication and draft saving are necessary for the system to be used, as also indicated in Chapter 4.1. Being in a corporate environment, it is also necessary to provide accountability which could be enabled by providing authentication. User drafts also pose merit for the TTTM system as they can enable other user features, for example, with draft saving the user will be able to send a Test Step suggestion to developers and save the current Test Case as a draft until the step is implemented into the Automation Test Framework.

Table 4.6: An example of User Features

4.7 Requirement specification

This chapter aims to summarize the results of the conducted analysis into a set of system requirements categorized based on the model in Chapter 2.4. Before the list of the requirements, this chapter also presents the list of user available actions in the form of use case diagram. These actions were inspired by the Low-Fi prototype used for user testing in Chapter 4.6. The main functionality of the system is then described as the most crucial information flow of the TTTM system. This flow is also based on the Low-Fi prototype with extensions derived from the sub-questions of the project's research question from Chapter 1.2.

4.7.1 User available actions in TTTM

The actions available for users in the TTTM system are shown in a graphical representation via Use case diagram Figure 4.11. The solution includes only one user - the Test Case Manager as defined in the delimitations of the project in Chapter 1.3.

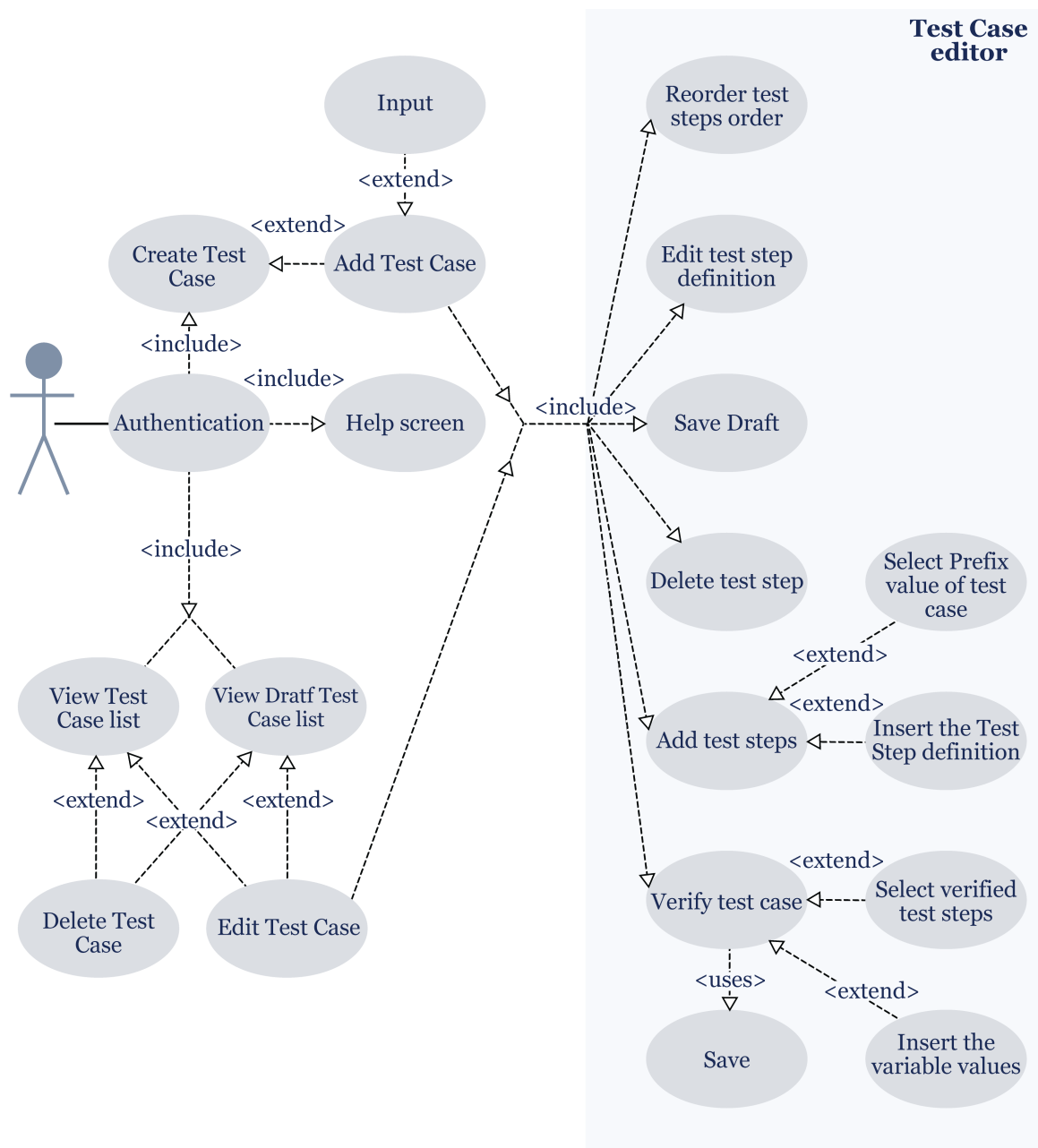


Figure 4.11: Use case diagram, describing TTTM Architecture.

First of all, to be able to operate in TTTM, the user must authenticate. Afterwards, the user can view the existing Test Cases and the "Help" page that provides explanations and guidance to the TTTM system. While viewing the existing Test Cases, the user can choose to delete them, edit or create new Test Cases. Upon creating a new Test Case, the user is required to enter its name. Later, the user is forwarded to the Test Case Editor page. This same page can be accessed if the user decides to edit an existing Test Case.

In this page, the user can create Test Steps, delete existing Test Steps, reorder the existing Test Steps, edit existing Test Steps, save a draft of the Test Case and verify the Test Case. Upon creating new Test Steps, the user is required to insert their textual input and choose a BDD prefix. The user is able to save the Test Case as a draft at any time while being in the Test Case Editor page. If the user performs a Test Case verification, the system presents a Test Step recommendation to the user. They can then choose the most suited Test Step out of the suggestions and add the required variables to it. Finally, when the Test Case is successfully verified, the user can save it.

4.7.2 The main information flow of the TTTM system

The main information flow of the TTTM system is represented visually in Figure 4.12. It specifically aims to define the preliminary flow of the creation of a new Test Case by using MERN technology stack described in Chapter 4.2. The visualized flow is shown in a form of a Sequence diagram.

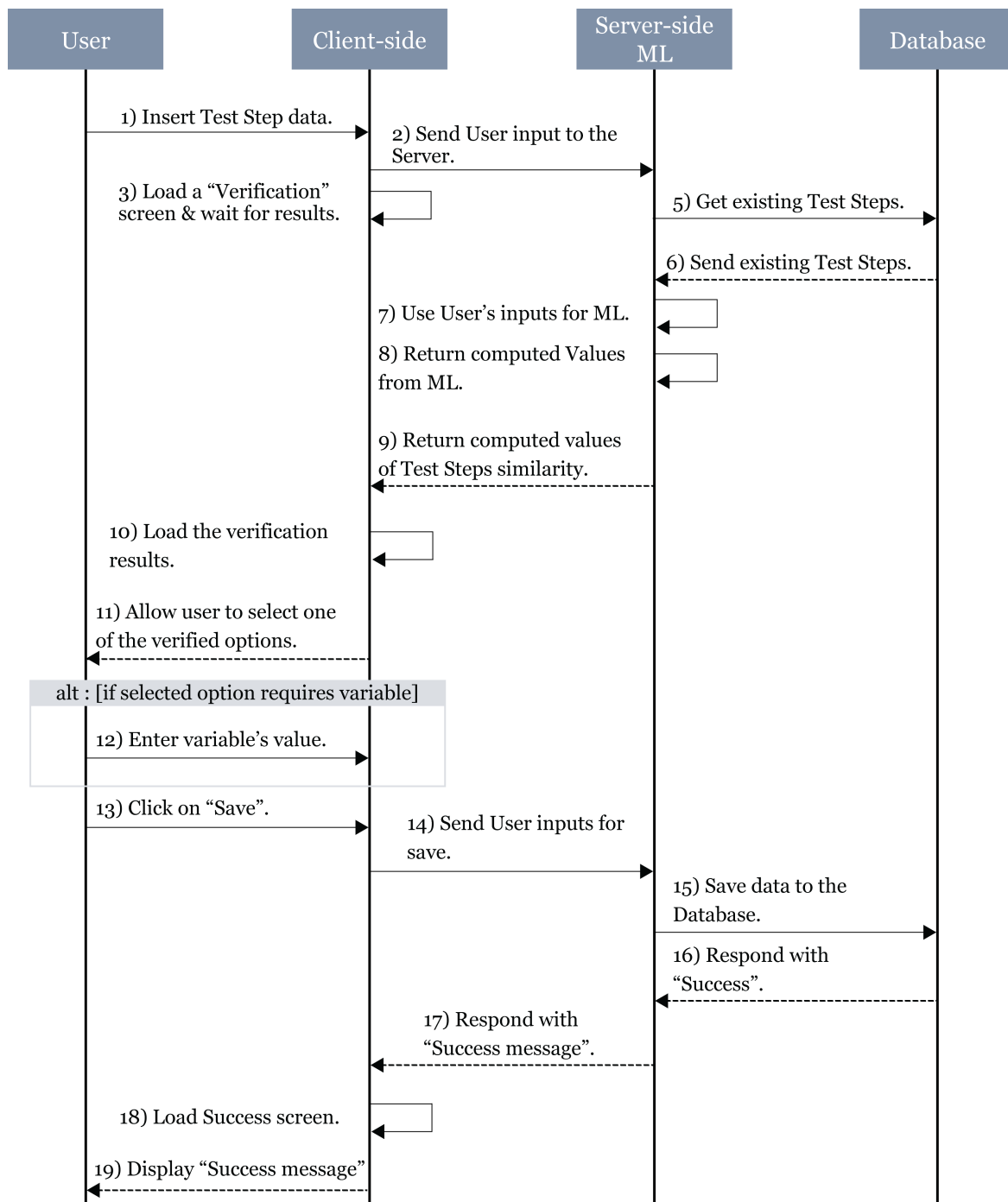


Figure 4.12: Sequence diagram, describing TTTM communication on Test Case creation.

The Figure 4.12, consists of four entities - User; Client-side (Web application); Server-side/ML (Back-end) and a Database. The flow is initiated by a User when the request of adding a new Test Case is sent out. To which Client-side responds with opening a creation page. Then User has the opportunity to enter Test Steps and Prefixes next to it. After the "Compile" button is pressed the Validation process of the Test Steps is initialized. The Web application passes the User input data to the Server-side by using HTTP POST request, then loads the next page for "Validation" and waits for Server-side response with output data. While the Web application waits, the Server-side communicates with the Database and retrieves the already existing Test Steps from it. The Server-side then combines the user input and already existing Test Steps. This information is sent to the ML, which returns the computed values to the Client-side that displays it to the User. The User then has to select recommendation out of the generated Test Step selection. If the selected option contained any variables, then the user must enter their values before proceeding to "Save" the Test Case. However, after the User clicks the "Save" button, the Test Case data is passed to the Server-side and then placed in the Database as a new Test Case document. The flow ends by redirecting User to another page, where it is announced that the Test Case was successfully added.

4.7.3 Requirements

In this chapter requirements of this project's prototype are presented. The requirements are important for setting up the scene for the future design and implementation chapters. The full list that contains all functional (29) and non-functional (25) requirements towards the project solution is placed in Appendix J. The group of this project is aware that the number of requirements towards such or a similar project normally is larger, however, due to the limited time-frame and the size of the project, it was limited to the above-mentioned numbers. Below are presented 10 functional and 10 non-functional requirements to demonstrate the capability of deriving them.

Functional requirements

MUST	FR #1 - User must be Authenticated before using TTTM solution.
Systems area:	Database Server-side Client-side
Source:	Chapter: 4.6 - #URF12
Description:	The authentication process is needed for such reasons as enterprise access management, security, and certain TTTM base functionalities.

MUST	FR #2 - Client-side must be SPA.
Systems area:	Client-side
Source:	Chapter: 3.6, 4.2
Description:	Because of SPA this project would enhance the User experience and provide a better solution in terms of Server request minimization and application responsiveness.

MUST	FR #3 - The system architecture must be based on MERN Stack.
Systems area:	Database Server-side Client-side
Source:	Chapter: 4.2
Description:	MERN stack is a dynamic and scalable approach for developing applications in the same language.

MUST	FR #4 - The system communication must use JSON format for data exchange.
Systems area:	Database Server-side Client-side
Source:	Chapter: 3.6, 4.2
Description:	The entities of MERN stack uses JS as its coding language and JSON as its data format for exchanging information.

MUST	FR #5 - TTTM System must be able to create new Test Cases.
Systems area:	Database Server-side Client-side
Source:	Chapter: 1.4, 4.6
Description:	It is the main flow of the system where Users can add a new Test Case which can consist of numerous Test Steps and their BDD Prefixes.

MUST	FR #6 - TTTM System must be able to update existing Test Cases.
Systems area:	Database Server-side Client-side
Source:	Chapter: 1.4, 4.6
Description:	It is a crucial flow which provides users with the ability to fix mistakes in Test Cases, for example, by reducing or rearranging the Test Step placement.

MUST	FR #7 - TTTM System must be able to display existing Test Cases.
Systems area:	Database Server-side Client-side
Source:	Chapter: 1.4, 4.6
Description:	This flow provides Users with an overview of all saved Test Cases and their Test Steps that are stored in the database.

MUST	FR #8 - TTTM System must be able to delete existing Test Cases.
Systems area:	Database Server-side Client-side
Source:	Chapter: 4.6
Description:	This action allows Users to identify and remove the selected Test Case with its Test Steps from the database and general display page.

MUST	FR #9 - Server-side must be able to forward the User inputs together with existing Test Steps to the LSTM model.
Systems area:	Server-side Machine Learning
Source:	Chapter: 4.3, 1.4
Description:	The Server-side must be able to use Test Steps from user's input as input values of the LSTM model.

MUST	FR #10 - Server-side must be able to receive the similarity values between the user inputs and the existing Test Steps from the LSTM model.
Systems area:	Server-side Machine Learning
Source:	Chapter: 4.3, 1.4
Description:	After the LSTM would produce the results of similarity scores of Test Steps, the Server-side must be able to receive and read these results.

Non Functional requirements

MUST	NFR #1 - LSTM model must be designed and trained using Keras Python library.
Systems area:	Machine Learning
Source:	Chapter: 3.5, 4.4
Description:	Keras is used to reduce the development process complexity and enable fast prototyping of ML models.

MUST	NFR #2 - The database must store Test Cases - their Name and their Test Steps.
Systems area:	Database
Source:	Chapter: 1.4, 4.6, 4.7.1
Description:	The database collection needs to be created for storing the information related to the Test Cases.

MUST	NFR #3 - The database must store 16 Test Steps - provided by case-study company - GN ReSound.
Systems area:	Database
Source:	Chapter: 1.4, 1.3, 2.2.1.
Description:	The database collection needs to be created for storing 16 Test Steps provided by GN ReSound.

MUST	NFR #4 - The database must store User information - Credentials & Draft Test Cases.
Systems area:	Database
Source:	Chapter: 4.6.
Description:	The database collection needs to be created for storing information related to Users.

MUST	NFR #5 - All input/select fields in the Client-side must contain labels describing what data is expected to be placed in it.
Systems area:	Client-side
Source:	Chapter: 4.6
Description:	Labels are essential for UX and indication of which field should store what value.

MUST	NFR #6 - The Client-side must allow users to select one of four BDD values in the Prefix section.
Systems area:	Client-side
Source:	Chapter: 4.6
Description:	This requirement is based on BDD in GN ReSound where they use four Prefix values: GIVEN; WHEN; THEN; AND.

MUST	NFR #7 - Client-side must provide identification placeholders of what variable type is expected in the variable input field.
Systems area:	Client-side
Source:	Chapter: 4.6
Description:	These placeholders would provide Users with information what the system is expecting in a specific variable field.

MUST	NFR #8 - The Client-side must indicate active edit fields with visual feedback of mouse cursor change to "text selector".
Systems area:	Client-side
Source:	Chapter: 4.6, 3.2
Description:	Users must know which field is active for editing; therefore, visual feedback on changing cursors shape must be applied.

MUST	NFR #9 - The Client-side must have a "Help" page which explains the prototype's features and functionalities.
Systems area:	Client-side
Source:	Chapter: 4.6
Description:	Help page would contain specific categories with tutorials on how to perform certain flows in TTTM system, also a general overview of it.

MUST	NFR #10 - The Test Cases in the TTTM system are represented in a BDD form.
Systems area:	Database Client-side
Source:	Chapter 4.6
Description:	It would represent the Test Steps & Prefix in list of one liners, not in a table with many categories.

Chapter 5

System design

This chapter presents the design decisions of the TTTM system's structure and data flows. These decisions were based on previously established MUST and SHOULD requirements from Chapter 4.7.3. In further explanations, the requirements are referenced by their ID value. The chapter covers an overall data structure of the solution with its underlying functionalities. Then the different data exchange flows are shown to elaborate on the purpose of each TTTM system's entities, presented in Chapter 4 Figure 4.1. Finally, the functionality of the TTTM system's API endpoints is defined.

5.1 Solution overview

Based on the research question in Chapter 1.2 and created requirements, a version of the TTTM system was designed. It was decided to construct a class diagram that would be able to represent the different data structures in the system. This specific design is an interpretation of the TTTM system which shows a minimalistic approach covering only the necessary data - Users that are present in the context of GN ReSound in Chapter 1.1, Test Cases that are defined in research question in Chapter 1.2 and Test Steps which are defined by the design of TTTM system in Chapter 4.1. The full figure is available in Appendix L in the form of a class diagram. Below, Figure 5.1 shows a block diagram with relations between classes of the TTTM system. The design of the TTTM system data structures consists of seven different classes which are described below.

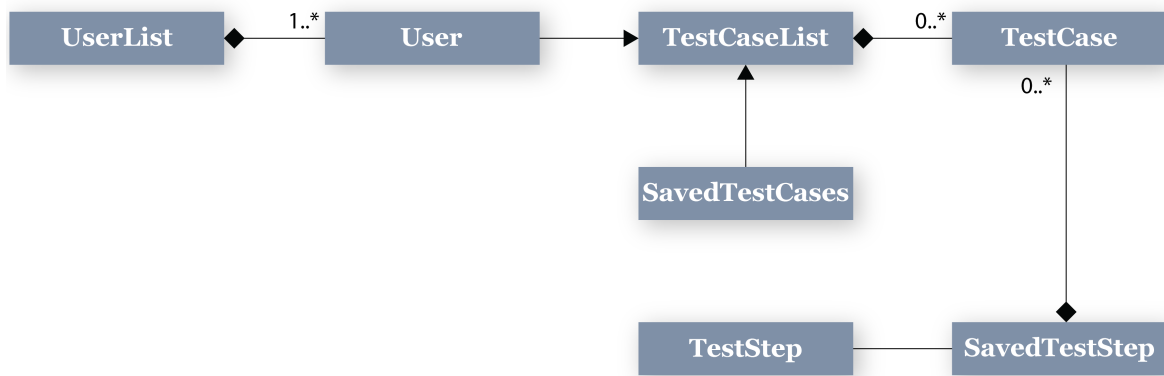


Figure 5.1: Classes in TTTM system

The underlying idea behind the design of the TTTM system indicates that the system would have multiple users that would be able to access it (FR #1). For this purpose, a **UserList** class was created which can be seen in Figure 5.2. This class consists of **UserObject**, which is an array of users. The system allows to add and remove users, which is an essential functionality for list structures. (**UserList**) contains at least one **User** class element. Each user has credentials in the form of a username and a password (NFR #14) and also a draftList attribute which consists of user-specific Test Case drafts (FR #14). The **User** class possesses "Set" methods that are used upon user creation, "Get" methods that allow retrieving user information (FR #7) and "deleteDrafts()" method that deletes all of the user's Test Case drafts (FR #8). The draftList is an attribute of **TestCaseList** class.

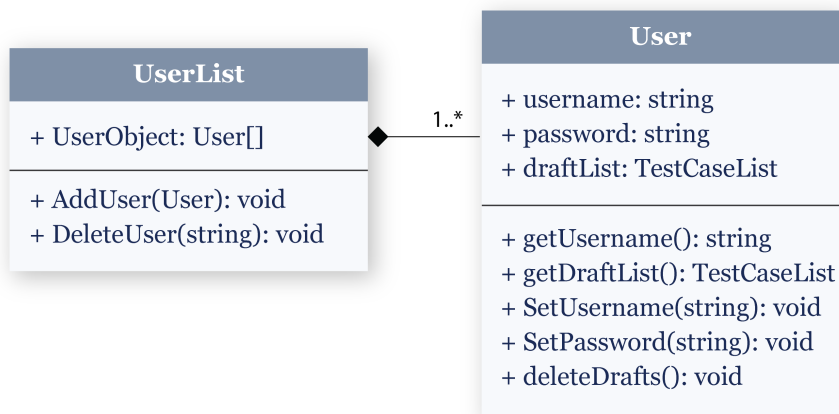


Figure 5.2: User related TTTM classes

Another instance of the (**TestCaseList**) class is presented in **SavedTestCases** class (NFR #12). This class consists of verified and saved Test Cases in the TTTM system. It enables the system to "Get" the verified test case list (FR #7) and to delete this list (FR #8). The **TestCaseList** class consists of **TestCase** class elements and is able to retrieve(**getTestCase()**)

(FR #7) or delete(DeleteTestCase()) (FR #8) a specific Test Case and add an additional one to the list(AddTestCase()) (FR #5). These Test Case list related classes can be seen in Figure 5.3.

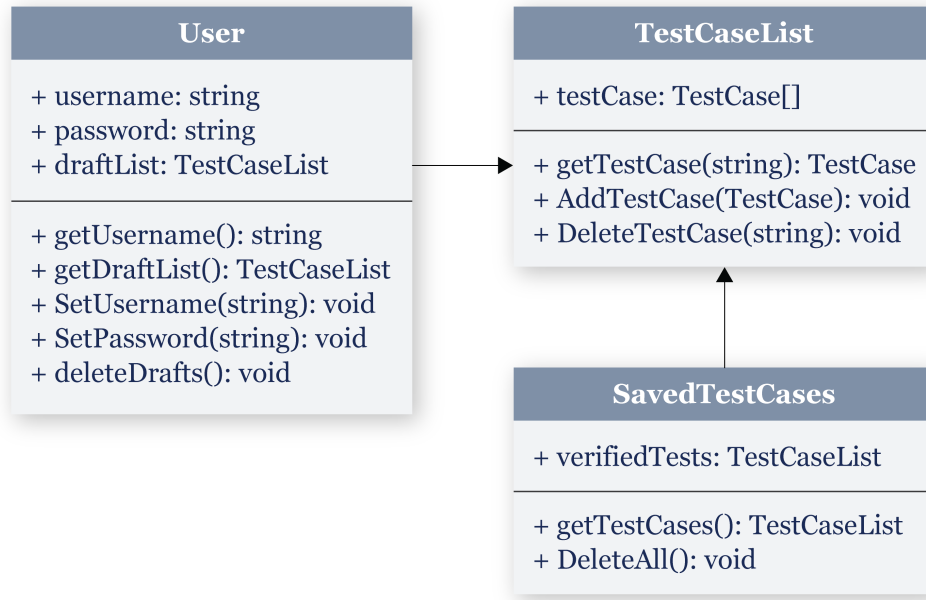


Figure 5.3: Test Case list related TTTM classes

A **TestCase** class consists of a Test Case name used for identification and a set of Test Steps using an array of **SavedTestStep** class (NFR #2). The **TestCase** class allows editing and retrieving its name, adding (FR #19) and removing of Test Steps (FR #24). Additionally, it is possible to get all the Test Steps of the specific Test Case and even change their order (FR #23). The **SavedTestStep** class consists of a Test Step definition and a State of the Test Step, indicating whether it was successfully validated or not (NFR #15) (NFR #16) (NFR #17). The class only possesses "Get" and "Set" methods for its attributes. These classes are presented in Figure 5.4.

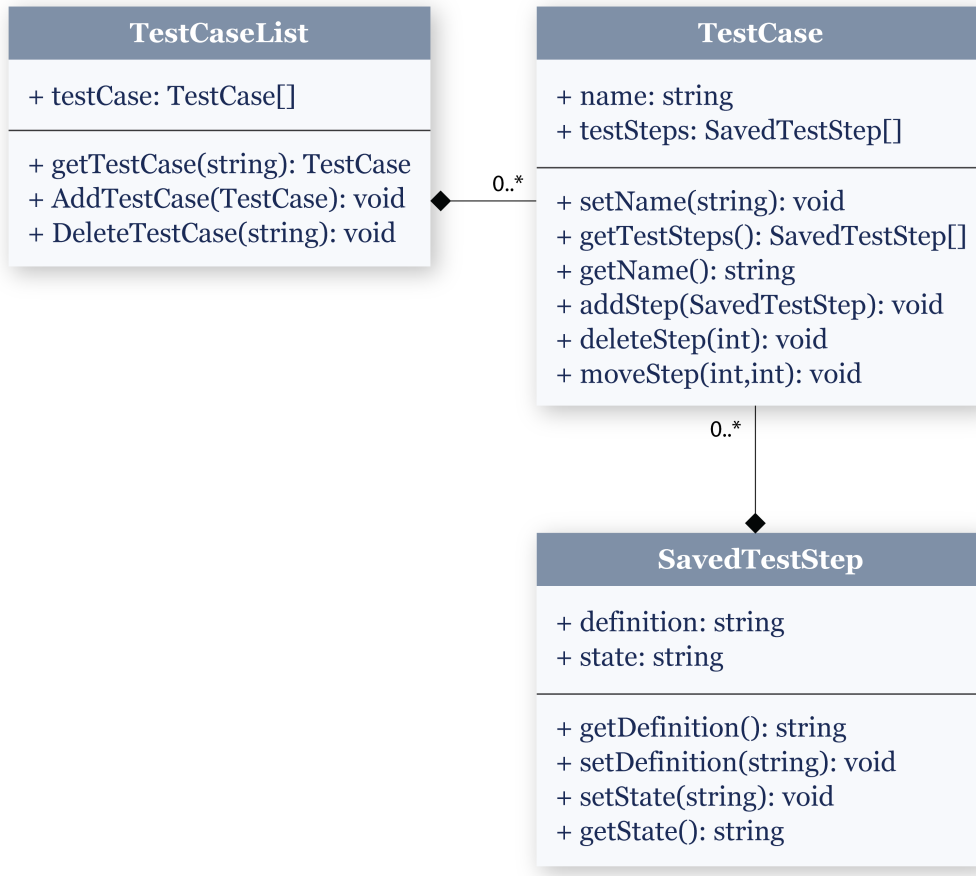


Figure 5.4: Test Case related TTTM classes

Up until this point, all of the presented classes are designed to be exposed to the end-user and give them access to underlying methods. The last class of the system is the **TestStep** class, which can be seen in Figure 5.5. It represents the "allowed" Test Steps against which user input is compared (NFR #3). Each **TestStep** element has a description that is the textual definition of the specific Test Step and an array of variable types that is present if required (FR #20). Apart from "Get" and "Set" functionalities for elements of this class, it is also possible to add and remove values from variable type arrays in situations when the Test Step's description is changed as well as retrieve all of the variable types. Lastly, the most important functionality `GenerateTestStep()` allows formulating Test Steps as string values of BDD format (NFR #10) that are used in **SavedTestStep** class. This method requires the description of the Test Step, the textual prefix defined by BDD structure (NFR #6), an array of the variable types and an array of variable values that would correspond to previously mentioned types.

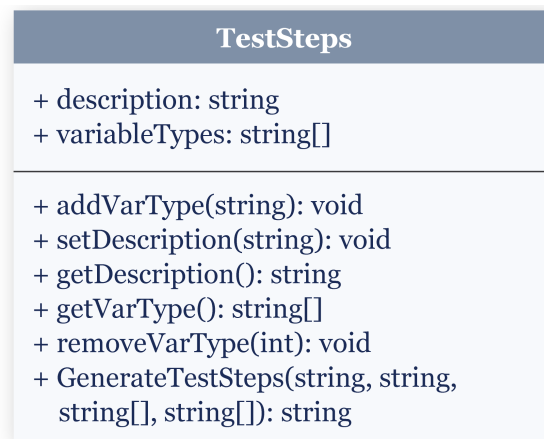


Figure 5.5: Test Step TTTM classes

5.2 System functionality flows

In this section, actions that users can perform in the TTTM system are defined and backed up by using visual UML Sequence diagrams for illustrating them in greater detail. Even though the entities in these diagrams are based on Figure 4.1 in Chapter 4 for the purpose of reducing complexity, ML and Server-side are shown as a combined entity. There are numerous functionality flows in the system; however, only the most important ones are described and illustrated below. The reason behind their selection is to reduce redundant and unnecessary information by only presenting CRUD actions that use different HTTP requests.

The most important actions available to the user are - Login, Add Test Case, Edit Test Case and Delete Test Case. Only after logging in and being authenticated the other actions become available for the user (FR #1). As defined in Chapter 1.3, the system was designed by taking in mind only one type of user, implying that none of the access management controls were considered. Errors are dealt with on both Client-side and Server-side, Server-side taking the responsibility of the majority of error handling and Client-side - UI error handling.

5.2.1 Login

Before gaining access to the TTTM system, an authentication process of the user is initiated. Without completing the authentication process, user has access only to the login and sign up page (FR #21). The procedure of authentication requires a Username and Password (NFR #14). In a full-scale implementation, these credentials should be the same as the enterprise credentials - provided by GN ReSound (NFR #20).

The authentication would be initiated when the user would provide the credentials and click on the "Login" button, as presented in Figure 5.6. This action would forward the user's entered credentials by using an HTTP POST request to the Server-side. Then the validation process would be performed. First, the Server-side would try to identify if the entered Username exists in the database. In the case of existence, the Server-side would compare the users entered password hash value to the one stored in the database (NFR #4). If both of these steps are valid, the Server-side sends a success message to the Client-side, which presents a user with his Test Case drafts (FR #7). In the case of an error in the validation flow, the Application would receive a response indicating that the user is unauthorized.

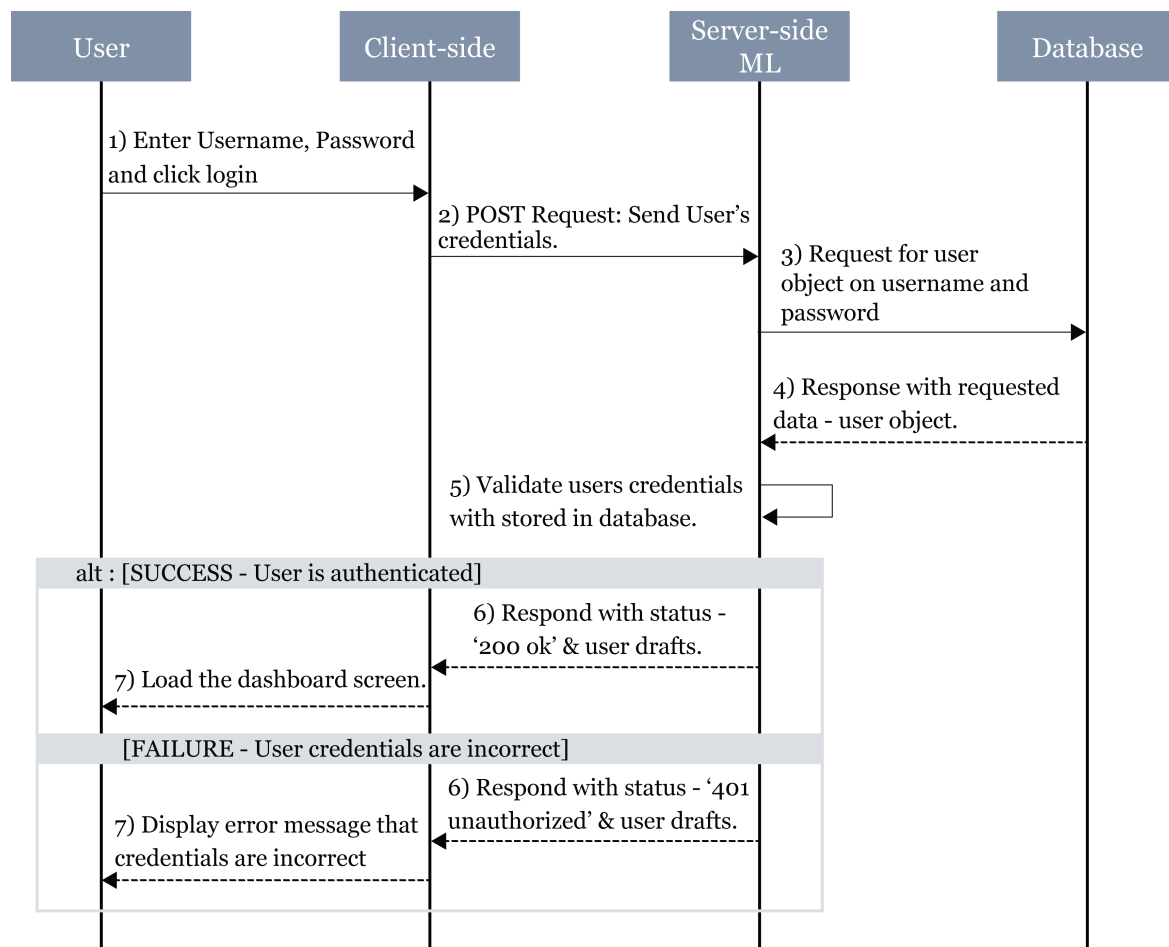


Figure 5.6: Login to the TTTM system

5.2.2 Add Test Case

The Add Test Case action is the main flow in the TTTM solution (FR #5), presented in Figure 5.7. It provides users with the ability to create new Test Cases by combining their inputs of Test Steps, Prefixes (NFR #10) and Test Case name (FR #22). This flow can be seen as an upgraded version of flow in Figure 4.12 located in Chapter 4.7.2, however, it is presented with greater amount of detail, particularly in the Server-side/ML entity.

The main adjustment of this diagram is located in the Server-side/ML entity where additional sequence points were added, and a few of the existing ones were redefined entirely. Particularly steps 12 to 14 contribute with a considerable update to the details of the diagram. Step 12 - Firstly creates a file, then exerts all stored Test Steps from the database, takes user inputs and places both in a relation file, mapping Existing steps to User input steps (FR #9). Step 13 - Secondly, the system runs the LSTM algorithm with the above-generated relation file, that computes similarity values for each pair of inputs in the relation file (NFR #13). Step 14 - The Web server receives the computed similarity values from LSTM model (FR #10). The remaining flow sequences are identical to the flow in Figure 4.12 that was explained in Chapter 4.7.2.

It is worth to mention that additional function of Adding Test Case Draft is available in the TTTM system. This flow is initialized whenever the user clicks on a button "Add as draft" at any point of the Add Test Case flow. After this button is clicked the steps 20 to 25 from Figure 5.7 are executed and a draft Test Case is saved to the user's collection.

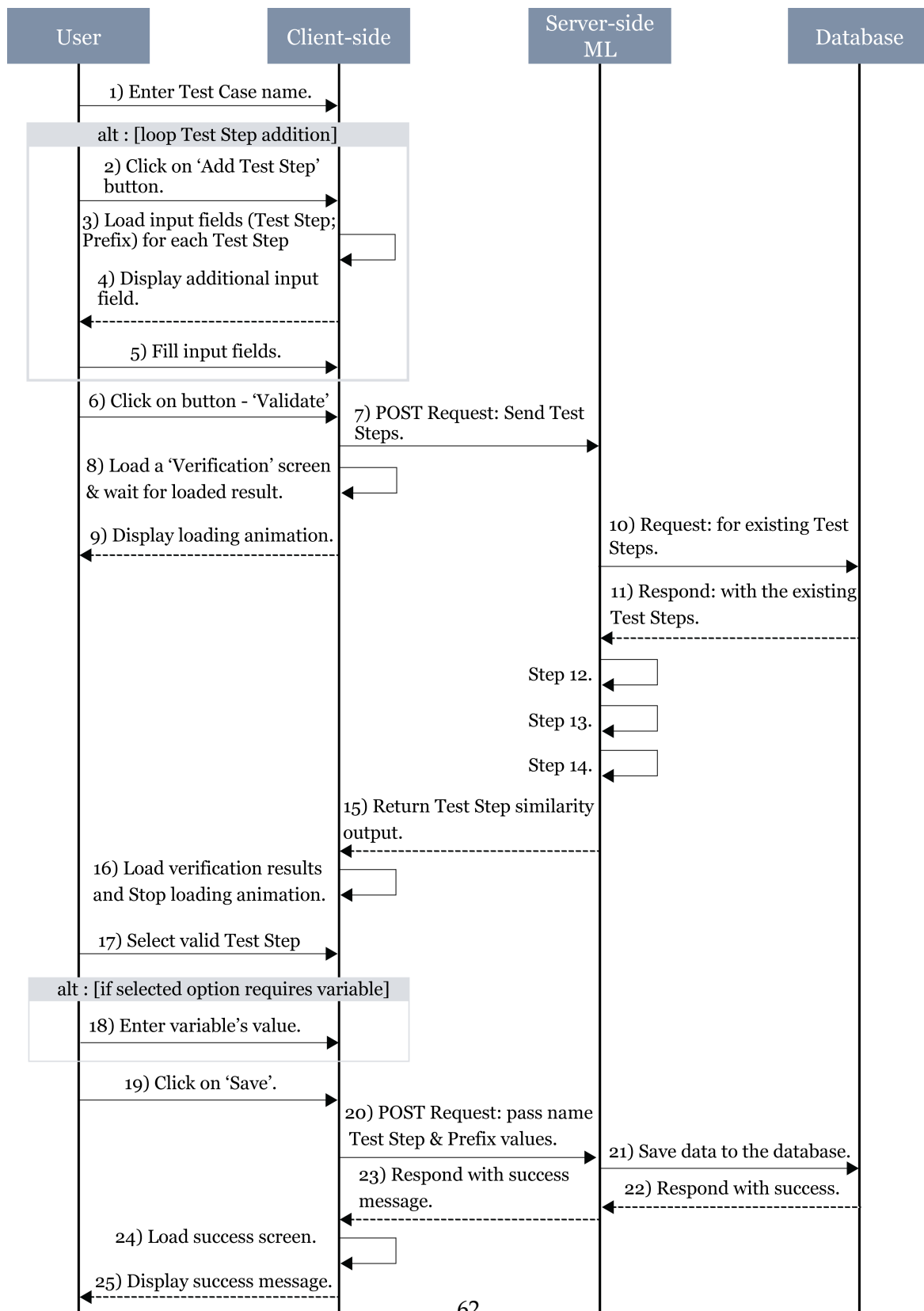


Figure 5.7: Add a Test Case of the TTTM system

5.2.3 Edit Test Case

Edit Test Case flow is reflected in Figure 5.8 below. The whole purpose of this action is to allow users to modify the content of the existing or recently created Test Cases (FR #6). The reasons might vary, e.g., replacement of old Test Steps, change of prefix values, reorder of Test Steps.

An edit action of the TTTM system is initiated when the user press on an edit button located in the List of Test Cases. After clicking on the button, an HTTP request is sent to the Server-side to prefill the form with an object of the corresponding Test Case's ID. The remaining flow is identical to the Create Test Case 5.2.2, from step 7 and up.

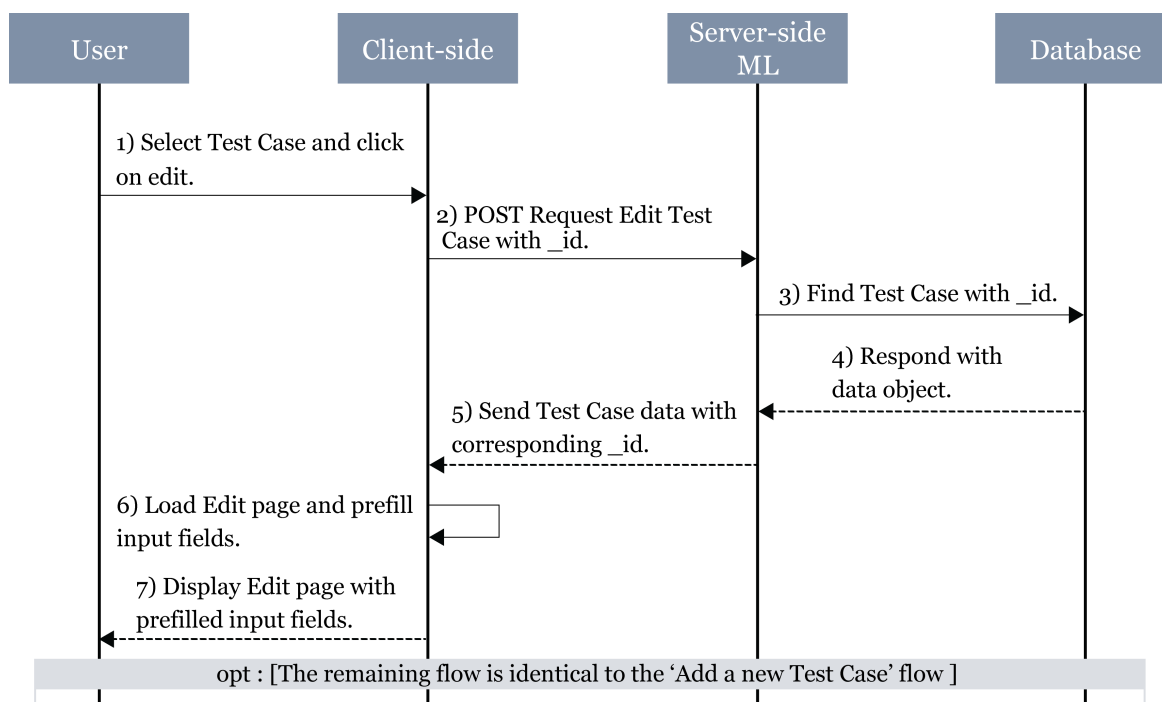


Figure 5.8: Edit Test Case of the TTTM system

5.2.4 Delete Test Case

The Delete Test Case flow is reflected in Figure 5.9 below. For deleting the Test Case user needs to go to the general list and click on the delete button. Such action would send the Server-side a DELETE request with an ID of the Test Case that should be removed. As a result, the Server-side would erase the required Test Case from the database and respond with the success message (FR #8).

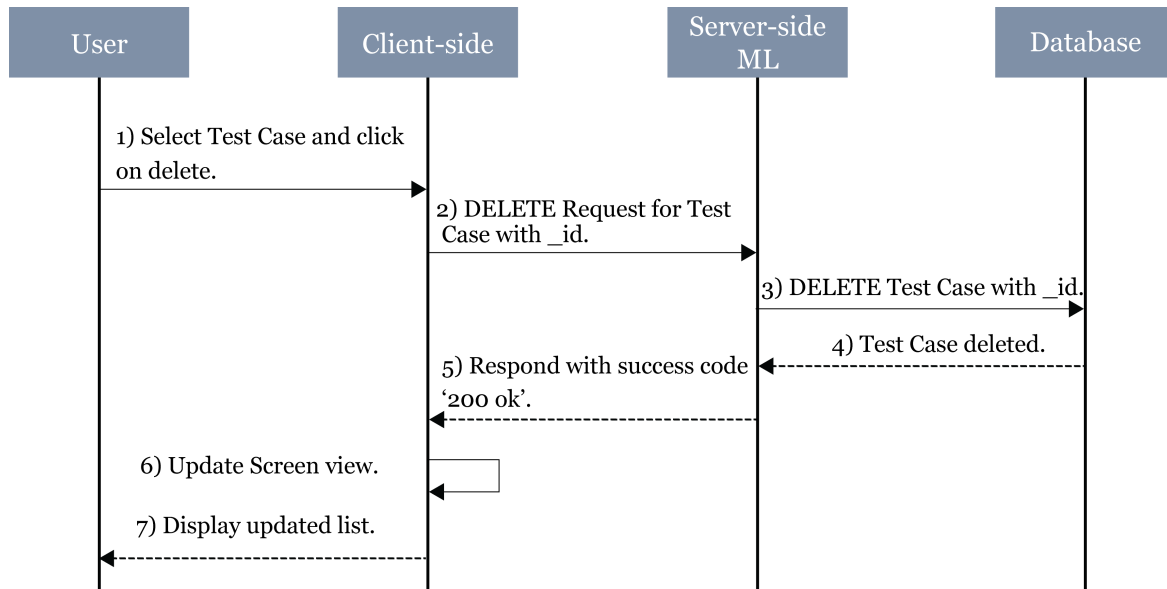


Figure 5.9: Delete Test Case of the TTTM system

5.3 API Endpoints

The TTTM system was designed so that Client-side and Server-side would communicate using API endpoints (FR #22). By combining information from the TTTM system's functionality flows, data structure and system requirements, a set of API endpoints were constructed. This chapter describes the purpose of each of these API endpoints and their basic functionality.

1. **GET: Test Case list** - This API is called from the Client-side in two different situations. It is initialized whenever the user attempts to view the Test Case drafts known as draftList in **User** class or the Test Cases that are available for all users and are ready for execution referred to as verifiedTests in **SavedTestCases** both presented in Chapter 5.1. After the request is received by the Server-side, it returns the list of Test Cases from the database based on the user's initially specified need. The Client-side shows the user a list of Test Cases with each having a list of Test Step definitions.
2. **POST: Login** - Login API endpoint is accessed on the user's authentication attempt. As described in Chapter 5.6, the Client-side transports user's username and password to the Server-side which verifies that data in the Database. Upon successful authentication, the Server-side actually calls **GET Test Case List** to present the user with their Test Case drafts.

3. **POST: Sign Up** - Before being able to access the TTTM solution, the user can sign up by providing a username and a password. This information is sent to Server-side and if the database does not contain such a username, it is saved as a **User** class element. Afterwards, the Server-side automatically uses the **POST: Login** API endpoint to authenticate the new user.
4. **POST: Save as draft** - This API endpoint is called whenever the user is in the middle of Test Case creation or editing process. It is initialized whenever user presses on corresponding "Save as Draft" button (FR #17). Afterwards, all of the Test Case data, including its name, Test Steps and their states are sent to the Server-side of the TTTM system. This Test Case is then saved in **User** Class draftList attribute introduced in Chapter 5.1. Server-side then responds to Client-side with a "success" message.
5. **POST: Validate** - This API endpoint is also accessed in the middle of Test Case creation and editing processes. It is a necessary step before user can save the Test Case for public accessibility. When this endpoint is called by pressing "Verify Test Case" button (FR #15), a list of Test Steps is sent to the Server-side. These steps are then verified against existing **TestStep** class element descriptions from the database as described in steps 12, 13 and 14 of Figure 5.7 in Chapter 5.2.2. The Server-side returns the Test Step list together with their similar Test Steps from database and their similarity scores. The state of each Test Step determines its color in the Client-side. Test Steps that are identical to the ones in database are marked green (NFR #17), the ones that are not similar to any - red (NFR #15) and the ones that had similar Test Steps in the database are yellow (NFR #16). The list of similar Test Steps also is attached to the yellow Test Steps (FR #12).
6. **POST: Save** - This API endpoint can be accessed from Client-side only when all of the Test Steps in Test Case creation or editing process are marked green which indicates that all of user defined Test Steps are validated (FR #18). After pressing "Save" button steps from 20 and onward in Figure 5.7 are performed. This Test Case is then stored in the Database as a list of strings representing its name and Test Steps as seen in **SavedTestStep** and **TestCase** classes from Chapter 5.1. Finally, the Server-side responds with a "success" message to the Client-side and the user is redirected to the list of Saved Test Cases by using the previously described **GET: Test Case list** API call.
7. **DELETE: Test Case** - The delete API endpoint request is initialized upon the User's performed action in the Client-side. It appears when the User presses a delete button of preferred Test Case that starts **DeleteTestCase()** method defined in Chapter 5.1. The Client-side forwards Test Case object's ID to the Server-side which verifies the sent ID with the ones stored in the database before performing delete request on the defined Test Case object. After the data object is deleted, the Server-side delivers an acknowledgment response to the Client-side which refreshes the test case list view.

Chapter 6

Implementation

This chapter presents the different implementation decisions that were made during the development of the prototype. The chapter offers the main highlights from both the Client-side and back-end development. In addition, the back-end subchapter covers the implementation of chosen Machine Learning algorithm that was required for deriving text semantic similarity.

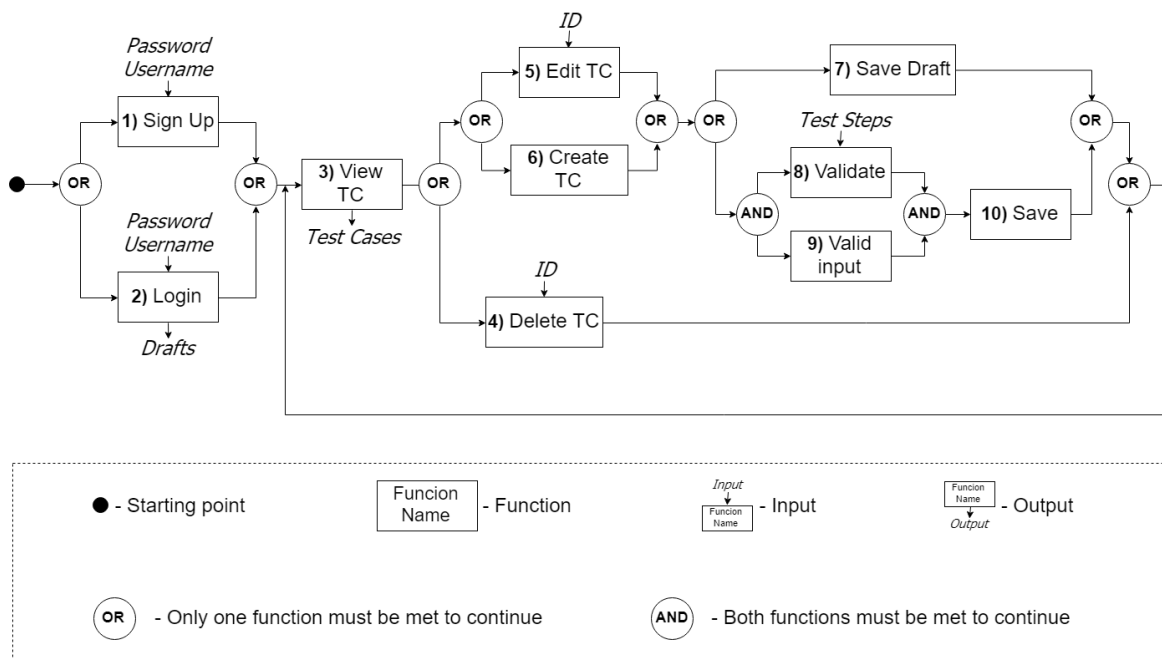


Figure 6.1: Functional diagram

Figure 6.1 above shows the function flow of the TTTM system. It is based on the TTTM system's class methods, information flows and API endpoints all described in Chapter 5. This figure introduces 10 function blocks in a sequential manner to show the overall flow

of the solution. The black dot on the left is the starting point for users upon launching the TTTM system. The blocks represent the system's functions, the text above blocks is the necessary input and the text below represents the output of that block. The system's functions are numerated to enable subsequent implementation chapters to reference them. The circles describe whether the user needs to pass both functions to continue the flow - AND, or only one function can be executed at a time - OR.

6.1 Data structure

In this chapter, the Database structure of the final TTTM solution is described. The Database contains three collections, that store different data Collections in a JSON format (FR #4). The first collection is Test Steps (NFR #3), the second is Users (NFR #4), and the last one is the Test Cases (NFR #2). Each data collection was designed with structure from Chapter 5.1 in mind, where class diagram fragments were defined for the TTTM system.

6.1.1 Test Steps collection

This collection holds Test Steps that were received from GN ReSound, particularly 16 (NFR #3) that are in the TTTM system's possession - this list is presented in Appendix D. The structure of the data object consists of three keys with their values, which are based on Figure 5.5 located in Chapter 5.1. **_id** as in any other data objects, is identifying each target with a unique "string" value that later on is used for identification before performing specific CRUD actions. **Step** key identified a string value of a Test Step that might include variables which are replaced with '(.*)' placeholders. As presented in Figure 6.2, steps can have more than one variable located in the same Test Step. **variableTypes** key represents an array that holds the type for each variable of the Test Step and is later used in the Client-side to present what is expected in the input field.

```
{
  "_id": {
    "$oid": "5c66a4a5e7179a27eb6134ec"
  },
  "step": "Change Volume to '(.*)' on '(.*)'",
  "variableTypes": ["number", "card"]
}
```

Figure 6.2: Test Step - JSON object.

6.1.2 User collection

The (User) data collection consists of the most important user information that is represented in three keys, excluding the object's **_id**, as seen in Figure 6.3. First two values are user credentials, where **Username** is a string and **Password** (NFR #4) is a hashed user's secret. Both of these key values are used in authentication flow as inputs, presented in Figure 6.3. This collection was based on the concepts from Chapter 5.1, Figures 5.2 and 5.3 and flows 1 and 2 from Chapter 5.2.1. In addition, this collection is used for Sign Up, Login, Create and Edit functions, displayed in Figure 6.1.

The remaining key is responsible for storing Draft related data, it is an array, and it is called **draftList** (NFR #4) similar to the definition presented in Chapter 5.1 Figure 5.3. This array stores objects that have an ID of the draft, the name of the Test Case and another array of objects that holds Test Steps which belong to the draft. The Test Step array structure includes four keys: **step**, **prefix**, **state**, and **variables**. The **step** and **prefix** values are strings that are BDD related. Moreover, the **state** value defines the state of the draft step in the validation process. This state can store "green" (NFR #17), "yellow" (NFR #16) and "red" (NFR #15) as values, which are specified in requirements of this project in Appendix J. The last key is **variable** which holds the type and the value of variables in the Test Step. The type is identifying what variable type is expected from user, and value for the user's input (FR #20) (NFR #7).

```

{
  "_id": {
    "$oid": "5cc03e564d463a2e3c61d547"
  },
  "password": "$2a@08@codsap11h82s2xGM8Lr0.Ov",
  "username": "TestUser",
  "draftList": [
    {
      "name": "Test Case Name",
      "_id": {
        "$oid": "5cc04d0564126528a0d07155"
      },
      "step": [
        {
          "step": "User is in Demo mode",
          "prefix": "Given",
          "state": "yellow"
        },
        {
          "step": "Volume set to '(.*)'",
          "prefix": "When",
          "state": "green",
          "variable": [
            {
              "type": "number",
              "value": "7"
            }
          ]
        }
      ]
    }
  ]
}

```

Figure 6.3: Users - JSON object.

6.1.3 Test Cases collection

The purpose of this collection is to store Test Case data generated by users in the TTTM system, as seen in Figure 6.4. Individually, each document data object contains **_id**, **name**, and **steps** keys (NFR #2). In addition, it stores the time value of when the object was **created**. The steps object is less complicated than the previously explained steps object in Chapter 6.1.2 and holds three keys - **_id**, **step**, and **prefix**. The design choices of this collection are presented in Chapter 5.1. This data collection is used for View Test Cases function shown in Figure 6.1.

```
{
  "_id": {
    "$oid": "5cc04d0564126528a0d07155"
  },
  "name": "Test Case Name",
  "created": "2018-02-20",
  "steps": [
    {
      "_id": {
        "$oid": "5cc04d05as21d8a0d07155"
      },
      "step": "User is in Demo mode",
      "prefix": "Given"
    }
  ]
}
```

Figure 6.4: Test Case - JSON object.

6.2 Client-side

For an overview of the Client-side understanding, Figure 6.5 is presented. It specifies the blocks and particular components of the TTTM solution. The flow starts at "Landing Page", where users have to either "Login" or "Sign up". After successful authentication the redirect to the root element called "Dashboard" is performed, where the user is allowed to use the system and navigate to the following application views - "Help", "Test Case Draft List", "Test Case List", and "Add Test Case".

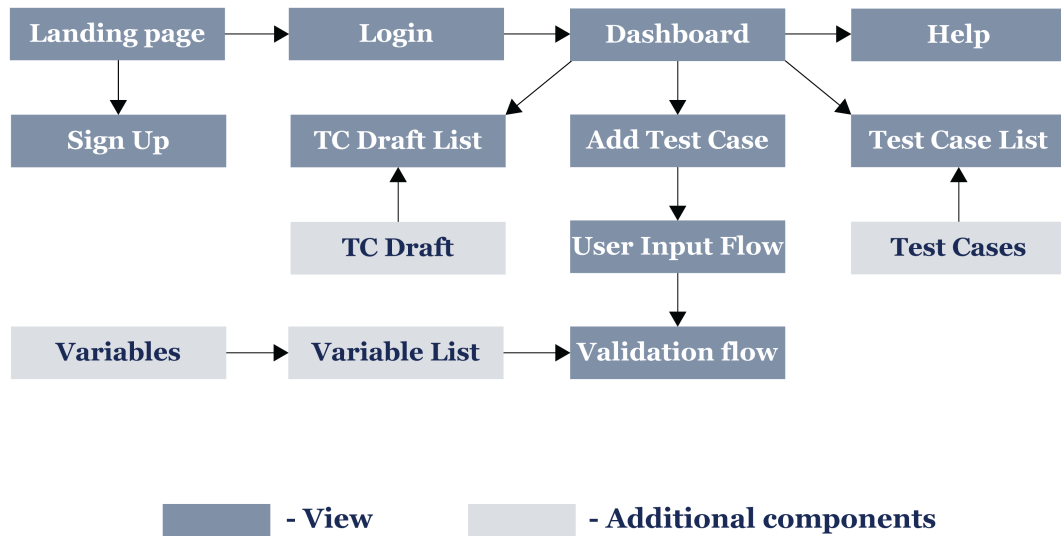


Figure 6.5: Client-side block diagram

All of these components but “Help” have additional child components that increase the functionality of the system. In particular, TC Draft and Test Case List views import the component that is responsible for listing the Test Cases together with its Test Steps. The “Add Test Case” consist of two flows - “User Input Flow” and “Validation Flow”. The first flow is responsible for getting the user’s inputs — the second for allowing users to select predetermined valid Test Steps and inset variables if there are any. Moreover, Figure 6.5 shows that “Validation flow” has few imported components that extend each other for the dynamic variable display.

In this chapter, the core code snippets that are important for Client-side of TTTM system will be described — starting with application data management, API calls, path routing, and Client-side GUI.

6.2.1 Managing application data

For in-app data flow, a state container manager called Redux [92] was used. This tool changes the ReactJS concept of working with data. Redux uses the central point for providing the data state into any application component. In this example, Redux would make sure that the whole data flow goes through the functions defined in Figure 6.1. This manager was chosen for reducing the complexity of data navigation in the React app by providing the following advantages for the app’s development: easy testing, debugging, and implementation of the data flows [92].

Figure 6.6 provides a code sample of how data is imported to the React component from Redux store. Afterwards, it is being mapped to the component and used inside of the component by creating the “destructured” [93] method to get access to the data values.

```

const mapStateToProps = (state) => ({
  masterForm: state.masterForm
});

//Destructure received data
const { name, testSteps } = this.props;

```

Figure 6.6: Get data to the component from Redux.

In order to update or perform any action, the application would call the action function which would dispatch the type and payload of it to the reducer, examples of such functions are discussed in Chapter 6.2.2 below. Then based on a function type that was executed, a case logic in reducer is initiated to update the state's value, Figure 6.7 provides an overview of multiple case statement logic.

```

switch (action.type) {
  case GET_TEST_CASES:
    return {
      ...state,
      testCases: action.payload
    };
  case DELETE_TEST_CASE:
    return {
      ...state,
      testCases: state.testCases.filter(
        testCases => testCases._id !== action.payload
      )
    };
  default:
    return state;
}

```

Figure 6.7: Switch and case logic for updating state of the Redux.

6.2.2 API Calls

In this chapter, Client-side API calls and their code snippets are presented. However, only API calls that do not share the same code base implementation will be listed below to provide with the most important implementation information. In addition, the aim of this chapter is to describe calls that utilize different HTTP CRUD requests. For the Client-side calls, "Promise based HTTP client" named Axios was used [94] which allowed creating HTTP requests with short and straightforward syntax on the Client-side JavaScript. The custom made HTTP requests were not considered essential for the prototyping. Therefore, such library was perceived as an enabler for the prototype's functionality development.

1. POST: Login

Figure 6.8 provides an overview of how an API POST request is initiated for Authentication reason (FR #1). It passes credentials of user's inputs to the Server-side where they get verified. As a result, Client-side waits for a response of the request, which identifies if the login was successful or not as shown in the Login function in Figure 6.1.

```
module.exports = app;
export const loginToTTM = (credentials) => async dispatch => {
  const res = await axios.post('http://localhost:4000/api/login',
    credentials
  );
  dispatch({
    type: AUTHENTICATE,
    payload: res.data
  });
};
```

Figure 6.8: POST request to the API for authentication.

2. GET: Test Case list

Figure R.1, which is located in Appendix R, is showing the implementation of the asynchronous function that uses GET request to receive the data from the custom made TTTM API endpoint (FR #22). This call is used for getting all saved Test Cases from the Database for the display purpose (FR #7). It is achieved by placing the received response into the "payload" variable that later is displayed on the Client-side, as seen in Chapter 6.1 function #3 View Test Cases.

3. **POST: Save - Test Case**

The implementation of this functionality is available in Appendix R Figure R.2 and is using the same API endpoint as "GET: Test Case list" request, however, it instead utilizes HTTP POST request. This function is responsible for the new Test Case creation (FR #5), in particular, it passes the object called testCases to the Server-side that stores the data in JSON structure as defined in Chapter 6.1.3 to the Database (FR #22), as seen in the Figure 6.1 function #10 Save.

4. **POST: Validate** The validation API also is using an HTTP POST request for dispatching a data object to the Server-side. The data object holds testSteps that are inputs of the TTTM system users for further use in ML. This flow is reviewed at greater detail in Chapter 5.2.2, as seen in Figure 6.1 function #8 Validate.

5. **DELETE: Test Case**

This API uses DELETE HTTP request, as seen in Figure R.3 located in Appendix R. Only an _id value is passed to the URL endpoint for pointing out the Test Case document that should be deleted from the collection (FR #8), as seen in function #4 Delete from Figure 6.1.

6.2.3 Routing of application

As React.JS helps to create a web-based SPA (FR #2), the routing on such app is accomplished by changing the URL endpoints and updating the view [95] [96] elements accordingly. Figure 6.9 provides the whole Route infrastructure that contains Paths and Components which are loaded on client request. In total TTTM solution has seven routes that open a different view category for users, for example, landing page or help page (NFR #9).

```

<Router>
  <div className="App">
    <Header />
    <Switch>
      <Route exact path="/" component={LandingPage} />
      <Route exact path="/test-case/add"
        component={AddTestCase} />
      <Route exact path="/test-case/edit/:id"
        component={EditTestCase} />
      <Route exact path="/test-case/list"
        component={TestCaseList} />
      <Route exact path="/test-case/draft-list"
        component={DraftCaseList} />
      <Route exact path="/help" component={Help} />
      <Route component={NotFound} />
    </Switch>
    <Footer />
  </div>
</Router>

```

Figure 6.9: Main App.js file that directs routing of application.

6.2.4 Client-side GUI

As the basis of the Client-side GUI, it was decided to use the previously designed Low-Fi prototype. The explanation of the Low-Fi UI elements and flows is available in Appendix H. The results from the user involvement proved that the majority of the flows and their GUI elements were acceptable for expert users. Additionally, their feedback allowed pinpointing the missing GUI elements and additional flow design in the TTTM solution. This chapter provides an overview of the final GUI of the prototype that includes six flows - Login, Sign Up, TC List, Draft List, Add TC, and Edit TC.

- **Add Test Case** - This flow proved to be accepted in terms of generated feedback from Low-Fi. Therefore, the GUI of this flow remained nearly identical to Low-Fi, as seen in Figure S.5 located in Appendix S. It stores TC name and allows to create or delete as many Test Steps with Prefix values as needed (FR #19) (FR #24). The Prefix has a range of four BDD selection options - Given, When, Then, And (NFR #6). In addition, new elements were added: labels next to each input or select fields (NFR #5) and the possibility to reorder Test Steps by dragging them vertically in the list (FR #23).

- **Validate Test Case** - This flow also remained nearly identical to Low-Fi and can be seen in Figure S.6 located in Appendix S. It included already defined color scheme of Low-Fi: red (NFR #15), yellow (NFR #16), green (NFR #17) for validation of Test Steps. In addition, the drop-down option selector for valid Test Steps was added (FR #12). Moreover, new UI elements - labels were added next to input or select fields (NFR #5), additional input fields per each variable in Test Step were generated (FR #13) and placeholders defining the type of the variable of the Test Step (NFR #7) were implemented. Finally, button namings were clarified, for validation - "Verify Test Case" (NFR #15); for saving - "Save" (NFR #17).
- **List of Test Case** - This GUI representation can be seen in Figure S.4 located in Appendix S. It provides users with all Test Cases stored in the TTTM system (FR #7), and their Test Step information (TS name, Prefix) in a BDD format (NFR #10). In addition, the deletion of Test Cases was enabled.
- **Edit Test Case** - This flow was accepted and verified by the expert users (FR #6). It does not differ much from the Add Test Case flow, so the UI for both of these flows is identical.
- **Draft List of Test Cases** - This flow was also created based on expert user feedback, and it can be seen in Figure S.3 in Appendix S. The GUI is similar to the List of Test Case flow, however, only displays the drafts of the specific user.
- **Login** - The GUI of the Login flow was not present in Low-Fi prototype of the TTTM system. Its creation was based on user demand to have the authentication feature. It consists of two input fields and a Login button, which allows users to authenticate with their Username and Password, as seen in Figure S.1 in Appendix S.
- **Sign Up** - This flow is created for enabling the Login flow by adding a user creation functionality. The GUI for it can be seen in Figure S.2 in Appendix S, also consisting of two input fields and a confirmation button.

6.3 Back-end

The back-end of the TTTM system consists of three elements first introduced in Chapter 4 and seen in Figure 4.1 - Server-side, Database and Machine Learning. The structure of the Database is already explained in Chapter 6.1, and in this chapter it is approached as an entity in data exchange flows. Figure 6.10 shows the function flow between the back-end systems. The arrows between blocks identify which way the information is sent. The Server-side handles all of the communications between these entities. ML Algorithm only exchanges information with the Server-side used for Test Step validation. From the three collections mentioned in Chapter 6.1, the Database exchanges Authentication information

and Test Case information with Server-side. Moreover, the Database provides the Server-side with Valid Test Steps for user input Validation purposes. In Chapter 6.3.2 below, Figure 6.14 shows a more detailed representation of these functionalities is presented in a form of API endpoints.

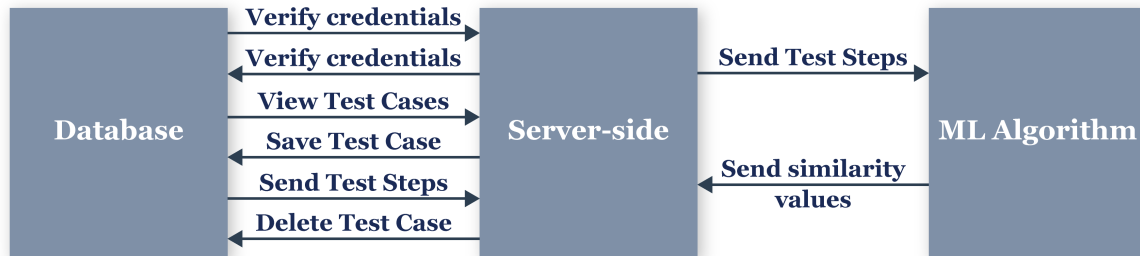


Figure 6.10: Main back-end functionalities

6.3.1 Machine Learning algorithm

The TTTM service employs an LSTM Neural Network model. This model is defined and trained using Keras tool(NFR #1). Figure 4.1 in Chapter 4 shows Machine Learning divided into the Model and Usage. This chapter refers to Usage component as the Neural Network's training file and to Model as the testing files and explains their implementation.

Training implementation

The training process of the LSTM Neural Network started with data set preparation. As described in Chapter 4.1, the inputs for the Machine Learning are textual pairs of Test Steps. The training set consisted of the Wikipedia data set used in Chapter 4.3.2 and additional 40 Test Steps collected from GN ReSound expert users in Chapter 4.5, which were used to create 400 new entries in the data set. Each of the pairs was given either a '1' or a '0' to determine if sentences are similar. These pairs are separated used as input to the *Word2Vec* word vector representation computing method (NFR #11). In essence, it uses a Google machine learning model to the data set and establishes a dictionary of word embeddings. This dictionary transforms all textual inputs to vectors of 300 [85] dimensions by applying 0-padding to data. Afterwards, the data set is split 9:1 for validation purposes. After this split, the LSTM model is defined in the Python script. As seen in Figure 6.11, the model had 50 hidden layers, 150 epochs and a batch size of 1024. The *Word2Vec* embeddings are then added to the model after which the hidden LSTM layers are appended. Next, the training input type is specified as number vectors. The output of the model is defined as the Manhattan Distance [86] between the inputs of the LSTM network. The model used a State of the art optimizer - Adam optimizer [97], which specified a learning rate of 0.001.

```

gpus = 1
batch_size = 1024 * gpus
n_epoch = 150
n_hidden = 50

# Define the shared model
x = Sequential()
x.add(Embedding(len(embeddings), embedding_dim,
               weights=[embeddings], input_shape=(max_seq_length,), trainable=False))

# LSTM
x.add(Dropout(0.2, input_shape=(60,)))

x.add(LSTM(n_hidden))

shared_model = x

# The visible layer
left_input = Input(shape=(max_seq_length,), dtype='int32')
right_input = Input(shape=(max_seq_length,), dtype='int32')

# Pack it all up into a Manhattan Distance model
malstm_distance = ManDist([shared_model(left_input), shared_model(right_input)])
model = Model(inputs=[left_input, right_input], outputs=[malstm_distance])

model.compile(loss='mean_squared_error', optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
model.summary()
shared_model.summary()

```

Figure 6.11: Definition of LSTM model

The actual training process was executed using *model.fit()* function that uses training data, batch size, epoch amount and validation data as entries. The resulting model had achieved a 0.85 training accuracy and a 0.81 validation accuracy, together with both training and validation model losses below 0.16, as seen in Figure 6.12. The figure shows a training process with 150 epochs, which can be identified as too many as the model achieved its optimal validation values on the approximately 30th epoch. Yet, the figure is presented this way to show that the validation error does not increase over time and overtraining does not happen. Even though these numbers were achieved in the training process, they are not guaranteed when applying previously unused user input Test Step definitions. The Machine Learning model testing using user Test Step inputs similarity is described in Chapter 7.1.

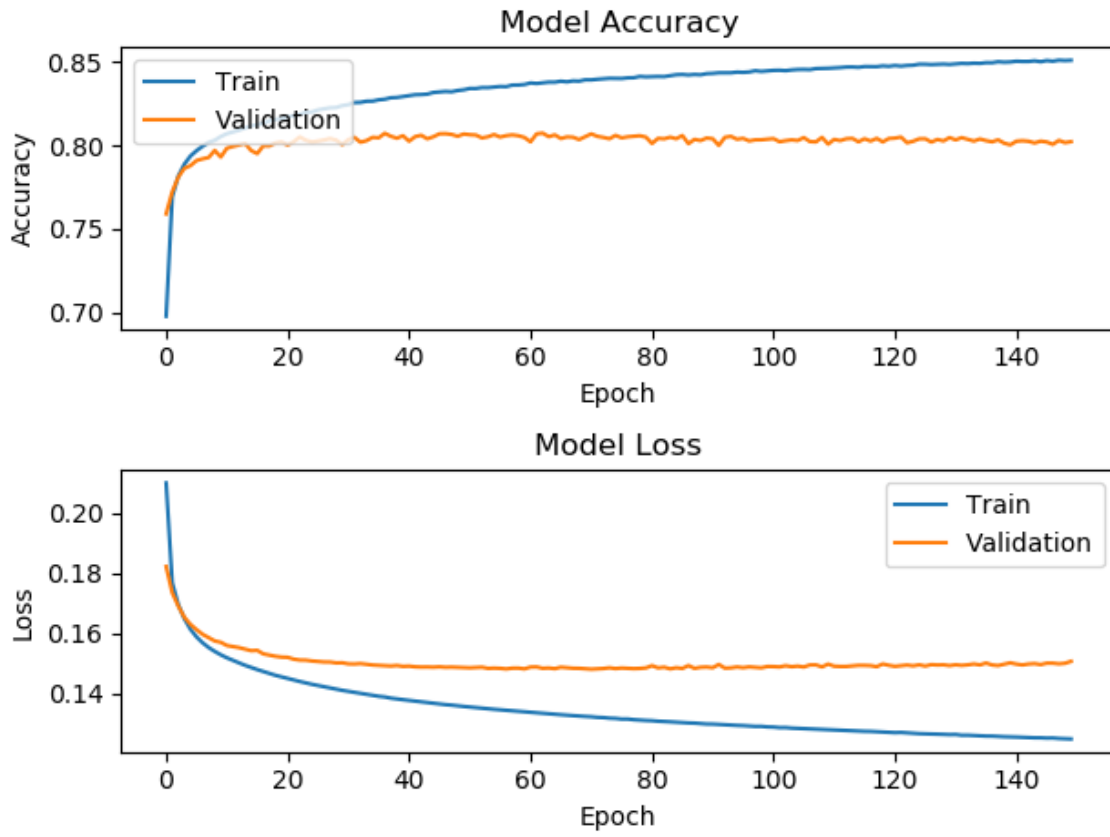


Figure 6.12: Accuracy and Loss of TTTM model

Model testing implementation

After finishing the training of the Neural Network, a Python file was created to execute the model within the TTTM Test Step validation process as described in Chapter 5.2.2. This model takes the previously created text file with user inputs and Database values as the input. The system applies *Word2Vec* to vectorize these textual inputs (FR #11). If previously unembedded words appear, the *Word2Vec* model is adjusted by adding the new embeddings. As explained in Chapter 4.3.2, the output of this model is defined in Manhattan Distance values of the input pairs. These values are then printed to the console. This overall process of the model usage is visualized in Figure 6.13.

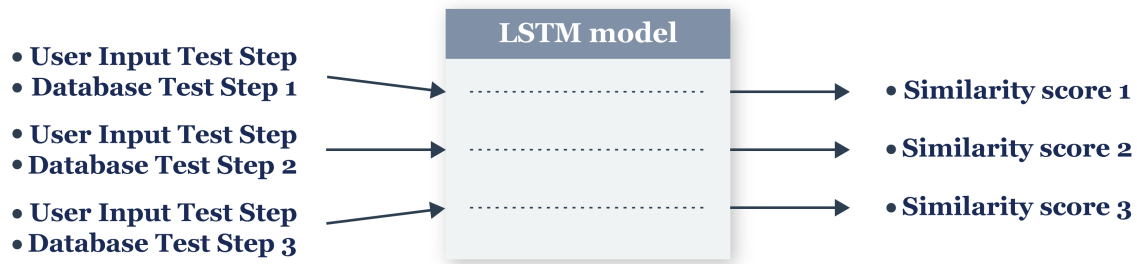


Figure 6.13: Model Testing implementation

6.3.2 Server-side

This chapter explains the implementation highlights of the Server-side of the TTTM system's back-end. The Server-side was written in JavaScript programming language in Node.js environment [98] as defined by MERN stack, which is used for the system's prototype (FR #3). The Server-side acts as the whole TTTM system's control unit that connects together the Client-side, Database and Machine Learning model. As specified by the MERN stack, the information exchange between these entities is done through JSON format files (FR #4). This chapter explains the implementation of API endpoints that are based on conceptualised functionality from Chapter 5.3. Figure 6.14 shows an overview of how the Server-side deals with back-end functionalities from Figure 6.10 and adopts the functions from the previously presented Figure 6.1. The functions in the figure below bare the same identification numbers and represent API end-points from Figure 6.1. The full figure with different sub-functions, inputs and outputs to the API end-points is available in Appendix P.

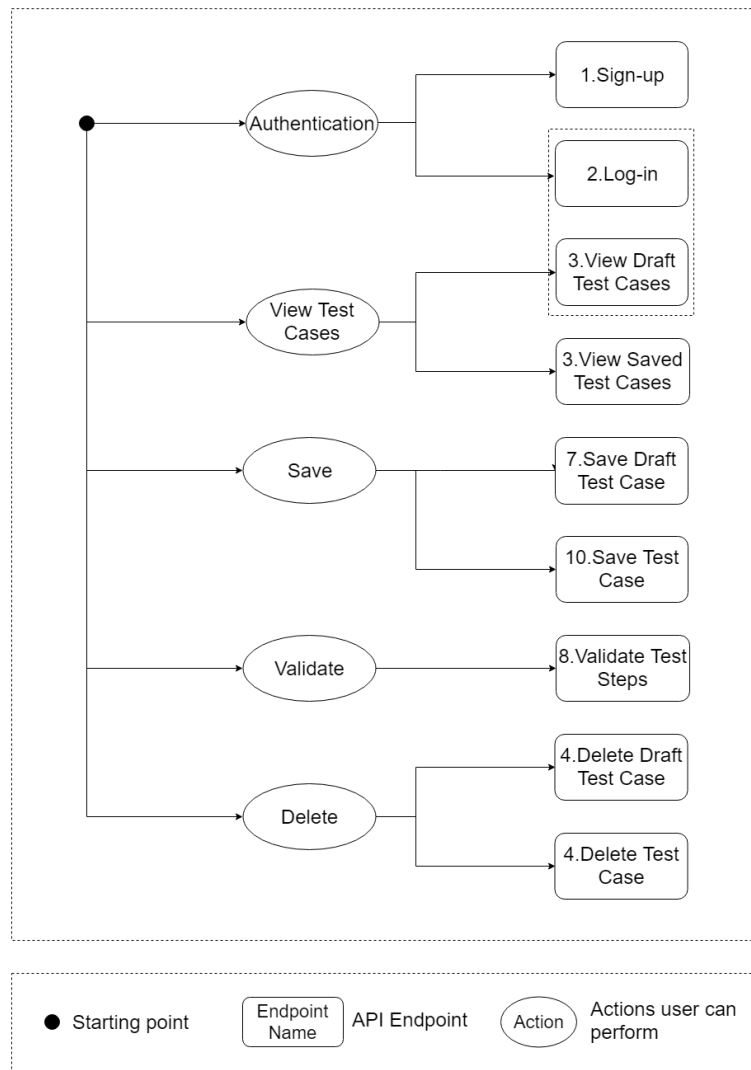


Figure 6.14: Server-side structure

Sign-Up implementation

The first functionality implemented in TTTM Server-side was the Sign-Up which served as an entry point for new users and in Figure 6.14, it is represented as function #1 Sign-up under Authentication(FR #1). It was implemented using Node.js Passport module [99] due to the Authentication not being considered a major contribution part of the project. Using Passports allowed for easy creation of Database user documents, which were enablers for prototyping of other TTTM functionalities, requiring user ID which all can be seen in Appendix P. It allowed to have two different flows depending on the outcome of actions. Upon success, the user is redirected to the main TTTM page, upon failure they are redirected back to Sign Up page.

```

passport.use("local-signup", new LocalStrategy(
  {
    usernameField: "username",
    passwordField: "password",
    passReqToCallback: true
  },
  function(req, username, password, done) {
    process.nextTick(function() {
      Users.findOne({ "username": username }, function(err, user) {
        if (err) return done(err);
        if (user) {
          console.log("no such user");
          return done(
            null,
            false,
            req.flash("signupMessage", "That username already exists.")
          );
        }
        else {
          var newUser = new Users();
          newUser.username = username;
          newUser.password = newUser.generateHash(password);
          newUser.save(function(err) {
            return done(null, newUser);
          });
        }
      });
    });
  }
));

```

Figure 6.15: Sign up implementation

Figure 6.15 shows that username and a password are requested for registering (FR #14) just as seen in Figure 6.1 and Appendix P function #1 Sign-up. The Database is then searched for an existing user with such a name. If the user already exists, as mentioned before, the Client-side is redirected back to Sign Up page and an error message is displayed. Upon success, the API endpoint saves the username and the hash of the user's password and redirects the user to the main page together with their user information.

Login implementation

Login functionality was also implemented using Node.js Passport module. The detailed explanation of the login flow is described in Chapter 5.2.1. The result of the Login API call is the same as the Sign Up so that upon success the user is redirected to the main TTTM page (NFR #14) and upon failure - to the Login page.

In the Appendix M Figure M.1, it can be seen that the input data from the user are the same as in Sign Up. First, the system checks if such a username exists. Upon finding the user, the Server-side verifies if the password hashes are identical. Upon success, all of the user information including the Test Case drafts (NFR #4) is sent to the main TTTM page. The merge between function #2 Log-in and the function #3 View Draft Test Cases in Figure 6.14 indicates that it was decided to retrieve user's drafts upon login to simplify the implementation. Upon any of the errors, the Server-side sends back to the Client-side an error message.

Test Case retrieval

This endpoint's functionality is to retrieve Test Cases from the Database. This endpoint retrieves saved Test Cases (FR #7) which is covering only one of the functionalities presented in **GET: Test Case list** API endpoint in Chapter 5.3. It is represented as the function #3 View Test Cases in Figure 6.14. The implementation is simple as the Server-side only needs to send a request to the Database to retrieve all of its Test Case entries, as seen in Appendix M Figure M.2. This functionality is designed specifically for the TTTM system's prototype. For the actual product, this approach would not prove being scalable. One of the solutions is to create pagination so that the Client-side would only receive a defined number of Test Cases specific to the request page of the list.

Test Step validation

Test Step validation is part of the Test Case creation process presented in Chapter 5.2.2. The Test Step validation is the single process that utilizes all of the TTTM system's elements in its execution. The endpoint is accessed when the user wants to validate the Test Steps in either editing or creation process of Test Cases.

```

exports.validate_test = function(req, res) {
  TestSteps.find( function(err, steps) {
    if (err) {
      res.send(500);
      return;
    }
    var RNNfile="";
    var result=[];

    for (var j=0;j<req.body.teststeps.length;j++)
    {
      for (var i=0;i<steps.length;i++)
      {
        RNNfile +=req.body.teststeps[j] + ", " + steps[i] + "\n";
      }
    }

    fs.writeFile("C:\\Users\\ezis94\\Desktop\\test.txt", RNNfile, function(err) {
      if(err) {
        return console.log(err);
      }

      console.log("The file was saved!");
    });
  });
}

```

Figure 6.16: First part of the validation process

The Figure 6.16 shows that the Server-side retrieves all of the existing Test Steps from the Database as seen in function #8 Validate from Figure 6.1 and the same function in the Appendix P. Afterwards, it creates a string that is populated with Test Step pairs that contains all of the possible combinations between user inputs and Test Steps from the Database (NFR #13). This information is then saved to a .txt file on the Server-side machine (FR #9). An example of a Test Step combination can be seen in Figure 6.17, where the left side consists of user inputs and the right side of the Test Steps from the Database.

```

Valite Demo mode is present, Verify that Hearing Aids are connected
Valite Demo mode is present, Swipe right on '(.*)' card
Valite Demo mode is present, Volume is changed to '(.*)'
Change volume to 6, Verify that Hearing Aids are connected
Change volume to 6, Swipe right on '(.*)' card
Change volume to 6, Volume is changed to '(.*)'

```

Figure 6.17: Example of Test Step text file

The second part of the Test Step validation endpoint executes the LSTM Sentence-based network model (FR #16). Figure 6.18 presents that after executing the Python file, the results are retrieved by catching the console output of the model (FR #10). These retrieved results are separated and the Test Steps with a similarity of 50% or more are compiled in a list together with user input they are verified against and sent back to the Client-side. Such number was chosen as the data set for the Neural Network had only zeros and ones as similarity values. If the outcome of Validation is more than 50% it means that the user Test Step is more likely to be similar to the Database Test Step, otherwise, it can be described as more likely to not be similar.

```
cmd.get(
  'C:\\Users\\Edgar\\Downloads\\lstm-siamese-text-similarity-master\\' +
  'lstm-siamese-text-similarity-master\\venv\\Scripts\\python.exe ' +
  'C:\\Users\\Edgar\\Downloads\\Siamese-LSTM-text-similarity-master\\' +
  'Siamese-LSTM-text-similarity-master\\predict.py',
  function (err, data, stderr) {
    var lines = data.split('\n');
    var StepArr=[];
    for (var i = 0; i < lines.length; i++)
    {
      if (lines[i].includes( searchString: "distance"))
      {
        lines.splice( start: 0, i);
        break;
      }
    }
    lines.splice( start: lines.length-1, lines.length);
    for (var i = 0; i < lines.length; i++)
    {
      console.log(parseFloat(lines[i].substring(11,lines[i].length)));
      if(parseFloat(lines[i].substring(11,lines[i].length))>=0.5)
      {
        console.log(i%steps.length);
        console.log(steps[i%steps.length].definition);
        StepArr = StepArr.concat({DBStep:steps[i%steps.length].definition,
          Vars:steps[i%steps.length].variableTypes,
          simVal:parseFloat(lines[i].substring(11,lines[i].length))});
        console.log(req.body.teststeps[Math.floor( % i/steps.length)]);
      }
      if (((i+1)%steps.length===0))
      {
        result = result.concat({userStep:req.body.teststeps
          [Math.floor( % i/steps.length)], stepsInfo:StepArr});
        StepArr=[];
      }
    }
    res.json(result);
  }
);
```

Figure 6.18: Second part of the validation process

Saving Test Case drafts

While creating or editing a Test Case, the user is able to save it as a draft, shown in Figure 6.14 as function #7 Save Draft (FR #14). This action saves the Test Case to user's account even if it is not validated. Figure M.3 in Appendix M shows both implementations - when a new draft is added and when an old one is updated. If the Client-side sends a draft that has an ID, then the Server-side searches for received ID and if found the old draft is replaced with a newer (FR #6). Otherwise, a new draft Test Case with its name and Test Steps is added to the user's draft list.

Saving Test Cases

The implementation of Test Case saving is also covering both creation and editing process available to view in Appendix P (FR #5). As seen in both Figure 6.1 function #10 Save and Chapter 5.3, Save request can only be sent after the Test Steps are validated on Client-side. In a similar manner to draft saving when the Server-side receives a Test Case with an ID, it checks Test Case collection in the Database to see if it exists. If such an ID is present in the Database, then the Test Case is overridden with the new one (FR #6). Otherwise, as shown in Figure M.4 in Appendix M, a new Test Case is created and added to the Database.

Test Case deletion

The implementation of deleting Test Cases covers both deletion of Test Case drafts and Saved Test Cases (FR #8). As seen in Figure M.5 in Appendix M if the request contains a user ID and a Test Case ID, the Server-side deletes the Draft Test Case entry from user's Draft List using the ID of the draft. If there is no user ID received in the request the Server-side deletes the Test Case from the common Saved Test Case collection in the Database. As seen in Figure 6.1 and Appendix P function #4 Delete Test Case, Test Case ID is the only mandatory parameter for the deletion. As the project does not delve into different user access control stated in Chapter 1.3, any user is able to delete from the common pool of Test Cases. Access control is a feature that would be implemented in the full-scale solution. Regular users would be able to only create Test Cases and only selected few - administrators of the system would be able to delete or edit existing ones. It can be done using the company access control of GN ReSound mentioned in Chapter 8.2, also enabling user action logging.

Chapter 7

Testing

Testing was performed at the end of the development phase of the TTTM solution to verify the system performance. As described in Chapter 2.5, there are three different testing methods used in this project - Machine Learning Accuracy testing, Black Box Software testing and User testing. All three methods were used to verify the implementation of the requirement list available in Appendix J. These requirements are referenced in the text similar to Chapter 6 and Chapter 5. Only a part of the requirement list was tested to not exceed the time resources of this project, therefore examples of different testing methods are shown in this chapter.

7.1 Machine Learning Accuracy Testing

The testing of Machine Learning module - LSTM neural network (FR #16) was conducted by sending Test Case, consisting of 17 Test Steps. 16 of these Test Steps were designed to serve the same meaning as the 16 Test Steps in the TTTM system's Database that can be accessed in Appendix D and one Test Step was designed so that it would not be similar to any of the stored steps (FR #9) (NFR #13). These steps can be seen in both Figure 7.1 and Figure 7.2. Inside the ML Algorithm, these textual steps were converted to numerical vectors using Word2Vec method (FR #11) (NFR #11).

After using these Test Steps as input to the LSTM network model the output showed that only seven Test Steps were similar to their counterpart in the Database and the 17th step was found similar to other Test Steps. In Figure 7.1 the resulting similarity values of the input Test Steps to their Database counterparts can be seen (FR #10).

1. "I press on Bottom Menu button",	0.22
2. "I swipe left to get to All-Around program",	0.81
3. "Swipe right to go to Outdoor program",	0.94
4. "Tap on Sound enhancer on Restaurant program",	0.70
5. "Both hearing aids are connected; left and right",	0.08
6. "I tap Tinnitus Manager",	0.36
7. "I press Music program on Top Ribbon Carousell",	0.05
8. "I press on Program Overview button",	0.64
9. "I tap Exit to Demo mode button",	0.47
10. "I set the left hearing aid volume to 8 in Music program",	0.46
11. "The right hearing aid volume is set to 2 in All-Around program",	0.35
12. "I split hearing aid volume bar in Restaurant program",	0.64
13. "I mute the merged streamer volume in TV1 program",	0.20
14. "I press Tips more menu item",	0.53
15. "I press on program card in Outdoor program",	0.29
16. "Scroll to the bottom of the page",	1.00
17. "I change Bass to -5"	

Figure 7.1: Results of the first NN accuracy testing phase

After reviewing the results, an improvement of the data set for Machine Learning was attempted. It was done specifically for the 16 Test Steps in the TTTM system's Database. A new interpretation was developed for each of them and then compared with all of the interpretations connected to these 16 Test Steps. In result, more than 3000 new Test Step pairs were added, both showing similar and not similar combinations. After retraining of the LSTM model using the updated data set, new results were gathered, seen in Figure 7.2. This time 10 out of 16 steps were deemed similar to their Database counterparts. Moreover, the model concluded that the 17th Test Step was not similar to any other Test Step from the Database. The validation accuracy of this model was also 0.81.

1. "I press on Bottom Menu button",	0.44
2. "I swipe left to get to All-Around program",	0.22
3. "Swipe right to go to Outdoor program",	0.67
4. "Tap on Sound enhancer on Restaurant program",	0.70
5. "Both hearing aids are connected; left and right",	0.005
6. "I tap Tinnitus Manager",	0.93
7. "I press Music program on Top Ribbon Carousell",	0.32
8. "I press on Program Overview button",	0.39
9. "I tap Exit to Demo mode button",	0.80
10. "I set the left hearing aid volume to 8 in Music program",	0.74
11. "The right hearing aid volume is set to 2 in All-Around program",	0.46
12. "I split hearing aid volume bar in Restaurant program",	0.77
13. "I mute the merged streamer volume in TV1 program",	0.77
14. "I press Tips more menu item",	0.77
15. "I press on program card in Outdoor program",	0.67
16. "Scroll to the bottom of the page",	1.00
17. "I change Bass to -5"	

Figure 7.2: Results of the second NN accuracy testing phase

It can be concluded that the TTTM system's Machine Learning part cannot provide the expected validation accuracy with inputs previously unseen to the model, however, it can be gradually improved by improving the LSTM model's training set, which would utilize the growing list of GN ReSound Test Steps for the full-scale solution. The summary table of performed tests can be seen in Appendix O.

7.2 Black Box Software testing

Chapter 5.3 described that the system has 7 functional API endpoints (FR #22), however based on the use cases presented in Chapter 4.7.1 and the implementation specifics in Chapter 6.3.2 it was realized that the Server-side of the TTTM system has ten functionalities - Sign-up (FR #1), Login (FR #1), View Test Cases (FR #7), Save Test Case (FR #5), Save Test Case Draft (FR #14), Delete Test Case (FR #8), Delete Test Case Draft (FR #8), Edit Test Case (FR #6), Edit Test Case Draft (FR #6), Validate Test Steps (FR #15). In total there were designed eight Black Box Software test cases, therefore, eight test scripts were created in Node.js framework, using JavaScript language. This decision was based on the fact that the View Test Cases and Login functionalities mainly provided the ability to retrieving the Test Cases or Test Case Drafts, therefore these functionalities were included into other test scripts.

Below in Figure 7.3, an example of such test script is visible for Validation functionality, where two Test Steps were used as inputs in a form of a JSON object (FR #4). This test script verified that these two Test Steps are then received back from the Server-side and that they both have corresponding similar Test Step from Test Steps in the Database shown in Appendix D (FR #12). The Test Steps were declared as similar if the LSTM model determined that they are at least 50% similar. Other seven test scripts are available in Appendix N. It is important to note that all eight were passed on first execution.

```

const request = require('request')
var assert = require('assert');

var toValidate={
  teststeps:[
    "I press Exit to demo button",
    "I swipe left to get to All-Around program"
  ]
}

request.post('http://localhost:4000/api/validate', {
  json: toValidate
}, (error, res, body) => {

  if (error) {
    console.error(error)
    return
  }
  console.log(`statusCode: ${res.statusCode}`)
  console.log(body)
  try {
    assert.equal(body[0].userStep, "I press Exit to demo button");
    assert.equal(body[0].stepsInfo[0].DBStep, "I press Exit to demo button" );
    assert.equal(parseFloat(body[0].stepsInfo[0].simVal)>=0.5,true );
    assert.equal(body[1].userStep, "I swipe left to get to All-Around program");
    assert.equal(body[1].stepsInfo[0].DBStep, "I swipe left to program program" );
    assert.equal(parseFloat(body[1].stepsInfo[0].simVal)>=0.5,true );

    console.log('The saving was successful.');
```

```

  } catch (error) {
    console.error('The saving was not successful.'+ error);
  }
})

```

Figure 7.3: Test script for Validate Test Steps endpoint testing

7.3 User testing

User testing was designed to enable answering the third sub-question from Chapter 1.2. The User Test participants were to execute test scenarios that were designed to provide an overview of the TTTM system's use cases presented in Chapter 4.7.1. These User Test Cases also validated the implementation of MUST and SHOULD requirements that are either related to the Use Cases or to the UI of the TTTM system from Appendix D. These User Test Cases are created in a form of tables that have the performed Test Steps, Expected outcomes and the IDs of relevant requirements seen in Table 7.1 below. The full list of the User Test Cases is available in Appendix Q.

1. Create		
Test Step	Expected outcome	Requirement ID
User opens the TTTM system	login page appears	FR #21
User presses on sign up	sign up page appears	
User enters a username and a password and presses submit	system redirects the user to his account page	FR #1, NFR #4, NFR #14
User presses on Create New Test Case tab	System redirects the user to the Test Case creation page	FR #5
User enters the test case name "New 1"	the name appears in the Test case name field	NFR #5
User adds 2 Test Steps	input fields appear	
User enters "Swipe left to get to Restaurant" in the first field and	requested text appears in input fields	
User chooses a prefix "When" for the first Test Step and "And" for the second Test Step	Requested prefixes appear near the Test Step edit fields	NFR #6
User attempts to press Save	Save button is disabled	FR #18
User presses Validate button	Both of the input field texts become yellow	FR #15, NFR #16
User presses on first input field selector	Test Step suggestions appear in a drop-down	FR #12
User selects "I swipe left to '(.*)' program" option	The input field text changes to the selected option, becomes green and an additional input field appears with "program" as a placeholder	FR #12, FR #13, FR #20, NFR #7, NFR #17
User enters "Restaurant" in the new edit field	The first Test Step input field changes to "I swipe left to 'Restaurant' program"	NFR #8
User presses on second input field selector	Test Step suggestions appear in a drop-down	FR #12
User selects "I set hearing aid volume to '(.*)' on right volume bar of '(.*)' program" option	The input field text changes to the selected option, becomes green and an additional input fields appear with "number" and "program" placeholder	FR #12, FR #13, FR #20, NFR #5, NFR #17
User enters "13" in the first and "Restaurant" in the second new edit field	The second Test Step input field text changes to "I set hearing aid volume to '13' on right volume bar of 'Restaurant' program" and the "Save" button becomes enabled	FR #13
User presses on "Save" button	User gets redirect to Saved Test Cases	FR #5, FR #17
User views the Test cases	"New 1" Test Case is present	FR #5, FR #7, NFR #2, NFR #12

Table 7.1: Test Case Creation User Test

Chapter 8

Reflections

The resulting TTTM system provides the world of Behaviour Driven Development with a web based tool which can increase the scalability and adaptability of BDD based testing frameworks. Specifically, it was developed utilizing the GN ReSound Automation Test Framework as the case study. The TTTM system utilizes Natural Language Processing to optimize the workflow between test automation developers and requirements writers by enabling a dynamic infrastructure for BDD based Test Case creation and automation. This infrastructure enables mapping of the undefined textual user input to the automation Test Step code functionalities.

To expand on the developed solution, this chapter presents discussions outside the scope of the current TTTM system's iteration. It starts with an evaluation of the overall project work together with encountered difficulties and possible improvements. It continues with a comparison between the full scale solution and the prototype system that was developed in order to solve the problem in Chapter 1.2 and to demonstrate the outcome of this project. Finally, future improvements of the project are presented, which include the potential systems expansion and increase in its capabilities.

8.1 Evaluation of project related processes

This chapter consists of a reflection on the TTTM project and its overall progress. It includes evaluation of the used process model, discussion of different encountered difficulties and possible improvements in the project work.

After comparing Scrum and RAD process models in Chapter 2.1, the project group decided to utilize the Scrum model for this thesis due to reasons that it fit both the report and prototype development of the TTTM system in an iterative manner. Specifically, it allowed for continues requirement formulation and narrowed technology selection based on the fast prototyping. The main advantage of using Scrum was that one of the group members was very tightly involved with the development of the Automation Test Framework in the case study company GN ReSound. This proved to be the key argument as

this group member would be proven to be a competent Product Owner for this project. However, if the focus would be initialized only on the development of service, the RAD model would fit this approach better due to continuous user involvement and feedback sessions in the planned sprints and overall development.

Throughout the project work, the group encountered multiple development related difficulties, which were then solved by utilizing the planned buffer of the time plan for the project. First of all, the development of the Client-side confronted multiple complications in TTTM feature development. They were caused because of the way the group decided to choose development web frameworks. This decision was based on the most compatibility for the TTTM system's purpose and the general performance of the web stack. The drawback of the decision was that group members were not familiar with the framework, so the Client-side development took more time resources than expected. As a result, two weeks out of the buffer time delegated towards the implementation and extra sessions were planed after the hand-in of the report. Another encountered difficulty was that the project aimed to analyze both simple machine learning algorithms and deep neural networks as candidates for the ML part of the TTTM system. As mentioned previously in Chapter 4.3.2, it is inefficient to compare machine learning algorithms only based on literature findings. For proper investigation, the results of each algorithm should be compared in the context of Test Step similarity detection. The group did not have time resources to perform such in-depth analysis. Instead, textual research of the algorithms in the field of text analysis was performed only two allegedly best algorithms were tested with preliminary data sets. Lastly, the gathering of the data set for the chosen ML algorithm was proven to be challenging. As GN ReSound is only in the starting phase of developing the Automation Test Framework, the amount of available Test Steps was limited. Moreover, different interpretations of the Test Steps were generated manually, which also was proven to be time inefficient. To mitigate the insufficient size of the data set, group member interpretations were added to the data set. After the hand in a sync with GN ReSound was planned to add any new Test Steps not present in the initial data set.

To improve the project work, the group came with several suggestions of what could be done differently to reduce crunch time and improve the productivity of the project. First of all, more time and strategies should have been applied in data set gathering. This could be done by dedicating more time for this task in the planning process. Moreover, other sources for the data set gathering could be used to train the ML system with more diverse inputs which are not directly relevant to Test Steps of mobile application Test Cases. Furthermore, more expert users could be involved in different times with more scenarios to test. Another improvement that would solve a previously mentioned challenge was to design the development phase in a more modular way. Upon finalizing the use of the specific web framework and the requirements of the system, the group could have designed features and elaborated on their implementation both from Server-side and Client-side perspectives. Afterwards, the shift-to-server-side should have been done, which would reduce the amount of work that was done in a less familiar React library, by redirecting

it to Node.js framework in which the group had prior knowledge. Lastly, a substantial amount was spent on training different neural network models to verify the most satisfactory combination of hyperparameters and proper data set. Even though this was process was not exceeding its planned time frame, it could have been minimized by using third-party services for the purpose of model training like Azure cloud [100] or AWS [101].

8.2 Complete solution vs prototype

As mentioned in the research question of this project in Chapter 1.2, the TTTM system was developed in regards to GN ReSound company. Because of the resource constraints of this project being a semester long, the provided prototype lacks features in comparison to a full-scale solution that could be adopted to the business of the case study company. The full-scale solution would be improved using the Could requirements from the full requirement list in Appendix J.

First of all, as GN ReSound is Microsoft business client and mostly uses .NET environment for its product development it would make sense to develop the ML part of the TTTM system using Microsoft Cognitive Toolkit (NFR #25). This would provide a higher level of employee involvement, smaller adoption period and tighter coupling with other company's products as the Software departments are already familiar with C# language that is the core of Neural Networks created with this Microsoft Cognitive Toolkit. Moreover, the validation accuracy of the TTTM system would need to be increased to be at least 88% (NFR #21) as described by the algorithm creators. This could be achieved by expanding the valid Test Step data set.

Other than using these Test Steps for Machine Learning, the TTTM system will also provide an interface for the developers so that they could expand on the valid Test Step list in TTTM system's database. This will allow the storage of the full list of the ones used in Automation Test Framework (NFR #22) and deletion of irrelevant and outdated Test Steps (FR #27). This feature would also require access control so that only specifically delegated employees would be able to interfere with the Test Step collection. It would make sense to use the already established company user credentials to keep the TTTM system up to the company's security standards (NFR #20). It would also enable user action logging.

As GN ReSound employees work in an agile environment, the requirements for their Automation Test Framework come from both code and requirement people. In a full-scale solution, the users of the TTTM system would be able to request new Test Steps to the developers if they see that current collection does not provide the needed implementation. This would be done through the UI of TTTM Client-side (FR #26).

The User Interface of the Client-side would also be improved based on the expert user desires described in Chapter 4.6. One of the biggest suggestions was to make the verification of the user input more dynamic. Upon writing Test Steps, the TTTM system would request similar valid Test Steps on each keystroke (FR #25). Even though, such functionality seems very appealing to the end-users, it would require GN ReSound to use

powerful machines for Machine Learning computations to minimize the amount of times the user would encounter response delays. Additionally, the Client-side would provide more guidance for the users in the Test Case creation process. As described in the BDD description in Chapter 3.1, "And" prefix Test Steps are connected to the previous Test Step. A visualization of this in the Client-side would give the user a better overview of their input (NFR #23). Finally, it was also requested by the GN ReSound employees to clarify the variable concept in the TTTM system. In the prototype, it is already done by presenting a type of which variable is requested from the user, e.g., "card", "item", "number". On top of that, each Test Step variable could have examples of the values that are expected - "All-Around", "About", "13", which would be stored together with Test Steps in TTTM system's database (NFR #24). Moreover, for usability purposes the color scheme for Test Steps in creation page could be supplemented with icons and mouse-over explanation to provide more information about the role of colors (NFR # 18) (NFR # 19).

8.3 Future improvements

Future improvements are the additions to the TTTM system that could improve its performance and create additional features both for the increase of the system's accuracy and usability. These future improvements were based on the **Won't** requirements presented in Appendix J.

8.3.1 Dynamic variable detection

In previous chapters, it is stated that some Test Steps in the GN ReSound Automation Test Framework have variables that the users can enter themselves. These are entries that are too specific to cover in Test Steps, so in the implementation of BDD, the framework can grasp where the variable is located in a valid Test Step. Hence, expert users suggested the TTTM system to dynamically perceive which of their inputs is the variable and where to put it in the validated Test Step seen in Chapter 4.7.1. This feature would need additional computational resources that would be used to determine which words of the textual input are to be treated as variables. This feature could be achieved by implementing another algorithm in the TTTM system (FR #28). This algorithm would be using the smartphone application for which the Test Case is being written. It would go through all of the UI elements of the application and compare each word of the textual input to these elements to determine which can potentially be a variable. Afterwards, while suggesting valid Test Steps to the user, the TTTM system would generate Test Steps with different variations of what is in the variable field based on the results of the new UI scanning algorithm.

8.3.2 Test Step addition by users

The expert users from the user involvement phase in Chapter 4.6 also suggested giving more control to the hands of the actual TTTM system users. The way it could be achieved

is by them being able to save their interpretations of Test Steps to the system. Such a feature could be added to the TTTM solution with some additions to improve its accuracy. The users would be able to choose an option to save their definition to the database of the system. This would enable to use their interpretation as a valid Test Step (FR #29). However, this is not the only implication of the new feature. These created definitions could also be forwarded to the Automation Test Framework at GN ReSound and added as valid definitions. Moreover, these new definitions would also be forwarded to the Machine Learning part of the TTTM system as additional inputs. This would potentially increase the accuracy of the LSTM model if it would be retrained after adding several new inputs.

8.3.3 Potential perspectives of the TTTM solution

While the TTTM system was developed with the idea of bridging manual and automated tests, it can have more implications on the enterprise industry. Apart from being used in Gherkin based systems, the redesigning of Test Cases using predefined Test Steps can increase the readability and validity of the company's documentation across different involved departments. By producing a unified definition, the reappearing Test Steps would not cause any confusion or imply any other meaning. It can also be said that such a system is a step towards creating a solution which would be able to create automated Test Cases by just analyzing the provided requirements.

Chapter 9

Conclusion

This thesis attempted to develop a system that delves into autonomous standardization of documented Test Cases, specifically, the convergence of textual Test Cases acquired from the case study company GN ReSound into test scripts ready for automation. Therefore, the problem formulation ended as the following statement:

How can test case documentation be used to create test scripts which can be executed autonomously?

Moreover, additional three sub-questions were defined in order to specify the scope of the project and the setup of the prototype. The answers to these sub-questions provided a detailed answer for the research question.

- *How can machine learning algorithms be utilized to support the dynamic creation of test cases?*

The project succeeded to introduce a solution that can utilize ML algorithms in the valid test case creation process. It was achieved by implementing a comparison functionality of existing and defined Test Steps with the newly created ones. This required the Machine Learning model to measure the semantic similarity between different textual inputs. In the presented solution, such model was developed based on LSTM Network and was able to achieve an accuracy of 81% for its predictions. It is important to note that this model only creates a set of suggestions out of which the user still needs to select a valid definition. Such solution reduces the documentation terminology knowledge that is needed for creating standardized Test Cases which in return provides a company's Test Case list with a defined structure and reduces the chance of human error. It also shortens the onboarding time for new employees in Test Case management processes. Moreover, this solution enables presenting Test Cases in a standardized form, so they can be executed in an automated manner, e.g., using BDD or TDD frameworks. Therefore, with this description, it can be concluded that this sub-question is answered.

- *How can a web application help users in the process of generating test cases?*

This question aimed to answer what features and functionalities should be developed in the TTTM solution. As a result, CRUD functionalities were implemented in the application to provide the users with the full control of their Test Case assets in the TTTM system - Create, Read, Update and Delete. These functionalities were enabled through a GUI which additionally provided its users with visual feedback when presenting valid Test Steps, suggested by the Machine Learning model. A Low-Fi prototype featuring these functionalities was presented to expert users of the case study company GN ReSound. The expert users identified additional features that would increase the usability and functionality of the system. These features included Test Step order manipulation in the Test Case and the concept of draft Test Cases, which would be available for later usage. After the refinement of these features, most of the user requests were added to the requirements and the prototype, hence providing an elaborate answer to this sub-question.

- *How can such a system be adopted in the context of the selected case study - GN ReSound?*

This question is essential to the project since this thesis aimed to optimize QA tasks in GN ReSound. For this reason, three expert users who are involved in Automation Test Framework development were interviewed. A Low-Fi prototype was presented to these participants to enable discussions and collection of the features desired by users. The final verification of the prototype was anticipated by creating scenarios based on requirements and performing user testing with expert users. However, because of scheduling issues, it was not possible to execute this phase with GN ReSound employees before the hand-in of this report. The results of this user test phase will be presented at the examination of this project. These user involvement phases serve as the answer to this research sub-question.

To conclude, it can be said that using the earlier mentioned research sub-questions, the main research question was answered by utilizing the research, analysis, and testing parts of this thesis. The project resulted in a working prototype that provides functionalities, which would optimize the use of BDD frameworks, and serves as a fundamental structure that can be expanded upon.

Bibliography

- [1] *SherWeb Comptia MSP Industry Outlook 2018*. Jan. 2018. URL: <https://bit.ly/2EwQEa4> (visited on 02/17/2019).
- [2] Raffi Margaliot Brad Little. *World quality report analyst report*. 2018. URL: https://www.microfocus.com/media/analyst-paper/world_quality_report_analyst_report.pdf (visited on 02/17/2019).
- [3] IEEE. *IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications*. 2011. URL: <https://standards.ieee.org/standard/830-1998.html> (visited on 02/07/2019).
- [4] Taisuke Hojo. *GHTF SG3 - QMS - Process Validation Guidance*. Jan. 2004. URL: <http://www.imdrf.org/docs/gh tf/final/sg3/technical-docs/gh tf-sg3-n99-10-2004-qms-process-guidance-04010.pdf> (visited on 02/07/2019).
- [5] Raffi Margaliot Brad Little. *World Quality Report 2018/2019*. 2019. URL: https://www.sogeti.com/globalassets/global/wqr-201819/wqr-2018-19_secured.pdf (visited on 02/07/2019).
- [6] Janet gregory Lisa Crispin. *Agile Testing: A Practical Guide For Testers And Agile Teams*. Dec. 2008, pp. 285–286.
- [7] Janet gregory Lisa Crispin. *Agile Testing: A Practical Guide For Testers And Agile Teams*. Dec. 2008, pp. 280–284.
- [8] *GN ReSound motivation*. URL: <https://www.resound.com/en-us/why-resound> (visited on 02/10/2019).
- [9] *Scaled Agile Framework – SAFe for Lean Enterprises*. URL: <https://www.scaledagileframework.com/> (visited on 02/07/2019).
- [10] *BDD: Learn about Behavior Driven Development | Agile Alliance*. URL: <https://www.agilealliance.org/glossary/bdd> (visited on 02/10/2019).
- [11] Dr. Jeff Sutherland. *Agile Development: Lessons learned from the first Scrum*. Oct. 2004. URL: <https://bit.ly/2KuzKwL> (visited on 03/03/2019).
- [12] H Mackay P Beynon-Davies C Carne and D Tudhope. *Rapid application development (RAD): an empirical review*. Dec. 2017. URL: <https://www.tandfonline.com/doi/pdf/10.1057/palgrave.ejis.3000325?needAccess=true> (visited on 03/02/2019).

- [13] *Boards* | Trello. URL: <https://trello.com/> (visited on 03/03/2019).
- [14] *ReSound Smart 3D hearing aid app*. URL: <https://www.resound.com/en/hearing-aids/apps/smart-3d> (visited on 03/30/2019).
- [15] Jacobijn A.C. Sandberg Maarten W. van Someren Yvonne F. Barnard. *The Think Aloud method - A practical guide to modelling cognitive processes*. 1994. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.821.4127&rep=rep1&type=pdf> (visited on 03/12/2019).
- [16] *InVision* | Digital product design, workflow & collaboration. URL: <https://www.invisionapp.com/> (visited on 04/11/2019).
- [17] *Databases and suppliers - Aalborg University Library (AUB)*. (Visited on 02/10/2019).
- [18] *Google Scholar*. URL: <https://scholar.google.dk/> (visited on 02/10/2019).
- [19] Joy Beatty Karl Wieggers. *Software Requirements*. URL: <https://books.google.lt/books?id=nbpCAwAAQBAJ%5C&printsec=frontcover%5C&hl=lt> (visited on 02/10/2019).
- [20] *BDD Overview : Cucumber*. URL: <https://docs.cucumber.io/bdd/overview/> (visited on 02/24/2019).
- [21] Dan North. *Introducing Deliberate Discovery*. Aug. 2010. URL: <https://dannorth.net/2010/08/30/introducing-deliberate-discovery> (visited on 03/15/2019).
- [22] *How to add Cucumber feature file in eclipse project*. URL: <http://www.automationtestinghub.com/add-cucumber-feature-file/> (visited on 02/10/2019).
- [23] *TextRazor - The Natural Language Processing API*. URL: <https://www.textrazor.com/> (visited on 02/19/2019).
- [24] *AYLIEN | Text Analysis API | Natural Language Processing*. URL: <https://aylien.com/> (visited on 02/19/2019).
- [25] *Wikipedia API*. URL: https://www.mediawiki.org/wiki/API:Main_page (visited on 03/15/2019).
- [26] *Dandelion API - Semantic Text Analytics as a service*. URL: <https://dandelion.eu/> (visited on 02/27/2019).
- [27] Osvaldo Simeone. *A Brief Introduction to Machine Learning for Engineers*. 2018, pp. 9–10. URL: <https://ieeexplore.ieee.org/document/8453245> (visited on 03/06/2019).
- [28] Taiwo Oladipupo Ayodele Yagang Zhang. *New Advances in Machine Learning*. Feb. 2010. URL: <https://cdn.intechopen.com/pdfs/10694.pdf> (visited on 03/06/2019).
- [29] Michael I. Jordan David M. Blei Andrew Y. Ng. *Latent Dirichlet Allocation*. 2003. URL: <http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf> (visited on 03/06/2019).

- [30] Michael W. Berry and Malu Castellanos. *Survey of Text Mining: Clustering, Classification, and Retrieval, Second*. Sept. 2007. URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.437.1442&rep=rep1&type=pdf> (visited on 03/07/2019).
- [31] etc. J.P. Yamron I. Carp. *A Hidden Markov Model Approach to Text Segmentation and Event Tracking*. URL: <https://bit.ly/2Ih2Ywg> (visited on 04/07/2019).
- [32] etc. Ryan McDonald Kerry Hannan. *Structured Models for Fine-to-Coarse Sentiment Analysis*. URL: <https://www.aclweb.org/anthology/P07-1055> (visited on 04/07/2019).
- [33] Marius Terblanche Adrian Georgevici. *Neural networks and deep learning: a brief introduction*. Feb. 2019. URL: <https://link-springer-com.zorac.aub.aau.dk/content/pdf/10.1007%5C%2Fs00134-019-05537-w.pdf> (visited on 02/26/2019).
- [34] *Artificial Neural Network : Beginning of the AI revolution*. URL: <https://hackernoon.com/artificial-neural-network-a843ff870338> (visited on 03/12/2019).
- [35] Charles Elkan Zachary C. Lipton John Berkowitz. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. June 2015. URL: <https://arxiv.org/abs/1506.00019> (visited on 03/09/2019).
- [36] Pranoy Radhakrishnan. *What are Hyperparameters and how to tune the Hyperparameters in a Deep Neural Network?* Aug. 2017. URL: <https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a>.
- [37] Liheng Xu Siwei Lai. *Recurrent Convolutional Neural Networks for Text Classification*. 2015. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9745/9552> (visited on 02/27/2019).
- [38] Junbo Wang Shiyang Liaoa. *CNN for situations understanding based on sentiment analysis of twitter data*. Dec. 2016. URL: <https://www.sciencedirect.com/science/article/pii/S1877050917312103> (visited on 04/11/2019).
- [39] Charu C. Aggarwal. *Neural Networks and Deep Learning*. 2018, p. 273.
- [40] Charu C. Aggarwal. *Neural Networks and Deep Learning*. 2018, pp. 39, 274–276.
- [41] Sepp Hochreiter. *The Vanishing Gradient Problem during learning recurrent Neural Nets and problem solutions*. URL: <https://www.bioinf.jku.at/publications/older/2304.pdf> (visited on 03/16/2019).
- [42] Charu C. Aggarwal. *Neural Networks and Deep Learning*. 2018, pp. 292–297.
- [43] Michael Nguyen. *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. URL: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21> (visited on 03/12/2019).
- [44] *TensorFlow*. URL: <https://www.tensorflow.org/>.
- [45] *PyTorch*. URL: <https://pytorch.org/> (visited on 03/15/2019).

- [46] *Features - Microsoft Cognitive Toolkit*. URL: <https://www.microsoft.com/en-us/cognitive-toolkit/features/> (visited on 03/19/2019).
- [47] *Home - Keras Documentation*. URL: <https://keras.io/>.
- [48] *Stack Overflow Developer Survey 2018*. URL: <https://insights.stackoverflow.com/survey/2018#technology> (visited on 02/20/2019).
- [49] *Welcome to the mean stack*. URL: <http://mean.io/> (visited on 02/20/2019).
- [50] IBM Cloud Education. *MEAN stack explained: A guide for getting started fast* | IBM. Sept. 2018. URL: <https://www.ibm.com/cloud/learn/mean-stack-explained> (visited on 03/01/2019).
- [51] *MERN v2.0 - Build production ready universal apps easily*. URL: <http://mern.io/> (visited on 02/20/2019).
- [52] IBM Cloud Education. *LAMP stack explained: Master the basics and get started quickly*. Sept. 2018. URL: <https://www.ibm.com/cloud/learn/lamp-stack-explained> (visited on 03/01/2019).
- [53] George Mihalas Antoine Geissbuhler Christian Lovis. *Connecting Medical Informatics and Bio-informatics*. 2005. URL: <https://books.google.dk/books?hl=en%5C&lr=%5C&id=HXTk5rW0dG4C%5C&oi=fnd%5C&pg=PA361%5C&#v=onepage%5C&q%5C&f=false> (visited on 02/27/2019).
- [54] *Introducing the MEAN and MERN stacks* | MongoDB Blog. URL: <https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack> (visited on 05/14/2019).
- [55] Gauge. *Open Source Test Automation Framework* | Gauge. URL: <https://gauge.org/index.html> (visited on 04/21/2019).
- [56] Cypress. *JavaScript End to End Testing Framework* | Cypress.io. URL: <https://www.cypress.io/> (visited on 04/21/2019).
- [57] Serenity. *Serenity BDD - Automated Acceptance Testing with Style*. URL: <http://www.thucydides.info/#/> (visited on 04/21/2019).
- [58] etc. Hrusikesh Panda Jess Chadwick. *Programming ASP.NET MVC 4*. URL: <https://www.oreilly.com/library/view/programming-aspnet-mvc/9781449321932/ch01.html> (visited on 05/04/2019).
- [59] Ioannis K. Chaniotis · Kyriakos-Ioannis D. Kyriakou Nikolaos D. Tselikas. *Is Node.js a viable option for building modern web applications? A performance evaluation study*. Mar. 2014. URL: <https://link-springer-com.zorac.aub.aau.dk/content/pdf/10.1007%5C%2Fs00607-014-0394-9.pdf> (visited on 04/03/2019).
- [60] *AngularJS Framework*. URL: <https://angularjs.org/> (visited on 02/20/2019).
- [61] *React – A JavaScript library for building user interfaces*. URL: <https://reactjs.org/> (visited on 02/20/2019).

- [62] Rajendra Banjade Vasile Rus Nobal Niraula. *Similarity Measures based on Latent Dirichlet Allocation*. 2013. URL: https://link.springer.com/chapter/10.1007/978-3-642-37247-6_37 (visited on 04/14/2019).
- [63] etc. David M. Blei Andrew Y. Ng. *Latent Dirichlet Allocation*. URL: <https://papers.nips.cc/paper/2070-latent-dirichlet-allocation.pdf> (visited on 04/14/2019).
- [64] Abhay Padda. *Introduction to topic modeling using LDA (Latent Dirichlet Allocation) - All About Analytics*. Oct. 2018. URL: <https://analyticsdefined.com/introduction-to-topic-modeling-using-lda-latent-dirichlet-allocation/> (visited on 04/14/2019).
- [65] etc. David Newman Arthur Asuncion. *Distributed Inference for Latent Dirichlet Allocation*. URL: <https://papers.nips.cc/paper/3330-distributed-inference-for-latent-dirichlet-allocation.pdf> (visited on 04/14/2019).
- [66] etc. Wen-tau Yih Kristina Toutanova. *Learning Discriminative Projections for Text Similarity Measures*. URL: <https://www.aclweb.org/anthology/W11-0329> (visited on 04/07/2019).
- [67] Chelsea Boling. *Semantic Similarity of Documents Using Latent Semantic Analysis*. Apr. 2014. URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.855.5331%5C&rep=rep1%5C&type=pdf> (visited on 04/09/2019).
- [68] *Fun with LSA (Latent Semantic Analysis)*. June 2004. URL: <http://www.robots.ox.ac.uk/~vgg/share/words/notes/toy4-doc.pdf> (visited on 05/10/2019).
- [69] Mounim A. El Yacoubi. *Hidden Markov Models for Text Recognition*. URL: https://www.researchgate.net/publication/282505340_Hidden_Markov_Models_for_Text_Recognition (visited on 04/07/2019).
- [70] R. Moore H. Murveit. *Integrating Natural Language Constraints into HMM-based Speech Recognition*. URL: <https://ieeexplore-ieee-org.zorac.aub.aau.dk/document/115777> (visited on 04/07/2019).
- [71] Zoubin Ghahramni. *An Introduction to Hidden Markov Models and Bayesian Networks*. 2001. URL: <http://mlg.eng.cam.ac.uk/zoubin/papers/ijprai.pdf> (visited on 04/07/2019).
- [72] etc. Manuele Bicego Vittorio Murino. *Similarity-based classification of sequences using hidden Markov models*. Apr. 2004. URL: https://profs.sci.univr.it/~bicego/papers/2004_pr.pdf (visited on 04/15/2019).
- [73] *Hidden Markov Model Implementation Module 'simplehmm.py'*. July 2005. URL: <http://users.cecs.anu.edu.au/~Peter.Christen/Febr1/febr1-0.3/febrldoc-0.3/node25.html> (visited on 04/15/2019).
- [74] Steve Young Mark Gales. *The Application of Hidden Markov Models in Speech Recognition*. 2007. URL: <http://mi.eng.cam.ac.uk/%5C~sjy/papers/gayo07.pdf> (visited on 05/09/2019).

- [75] Gernot A. Fink. *Markov Models for Pattern Recognition*. 2014. URL: <https://www.springer.com/us/book/9781447163077> (visited on 05/09/2019).
- [76] etc. Sebastien Marcel Olivier Bernier. *Hand Gesture Recognition using Input-Output Hidden Markov Models*. URL: <http://www.idiap.ch/resource/gestures/papers/marcel-fg-00.pdf> (visited on 05/10/2019).
- [77] Grzegorz Kondrak Wesley Mackay. *Computing word similarity and identifying cognates with pair hidden Markov models*. June 2005. URL: <https://aclweb.org/anthology/papers/W/W05/W05-0606/> (visited on 04/07/2019).
- [78] etc. Mangi Kang Jaelim Ahn. *Opinion mining using ensemble text hidden Markov models for text classification*. Mar. 2018. URL: <https://www.sciencedirect.com/science/article/pii/S0957417417304979> (visited on 04/07/2019).
- [79] Mihai Rotaru Paul Neculoiu Maarten Versteegh. *Learning Sentence Similarity with Siamese Recurrent Architectures*. Aug. 2016. URL: <https://www.aclweb.org/anthology/W16-1617> (visited on 04/04/2019).
- [80] Aditya Thyagarajan Jonas Mueller. *Learning Sentence Similarity with Siamese Recurrent Architectures*. URL: <https://dl.acm.org/citation.cfm?id=3016291> (visited on 04/04/2019).
- [81] Jimmy Lin Hua He Kevin Gimpel. *Multi-Perspective Sentence Similarity Modeling with Convolutional Neural Networks*. Sept. 2015. URL: <https://ttic.uchicago.edu/~kgimpel/papers/he+etal.emnlp15.pdf> (visited on 04/16/2019).
- [82] *Easy TensorFlow - Bidirectional RNN for Classification*. URL: <http://www.easy-tensorflow.com/tf-tutorials/recurrent-neural-networks/bidirectional-rnn-for-classification> (visited on 04/04/2019).
- [83] Isabelle Guyon Jane Bromley. *Signature Verification using a "Siamese" Time Delay Neural Network*. 1994. URL: <https://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network.pdf> (visited on 04/04/2019).
- [84] Mike Schuster and Kuldip K. Paliwal. *Bidirectional Recurrent Neural Networks - Signal Processing, IEEE Transactions on*. Nov. 1997. URL: <https://pdfs.semanticscholar.org/4b80/89bc9b49f84de43acc2eb8900035f7d492b2.pdf> (visited on 04/04/2019).
- [85] *Google Code Archive - Long-term storage for Google Code Project Hosting*. URL: <https://code.google.com/archive/p/word2vec/> (visited on 04/04/2019).
- [86] *Manhattan distance of a point and a line*. URL: <http://artis.imag.fr/~Xavier.Decoret/resources/maths/manhattan/html/> (visited on 05/11/2019).
- [87] *No Free Lunch Theorems*. Oct. 2012. URL: <http://www.no-free-lunch.org/> (visited on 05/07/2019).
- [88] *The Stanford Natural Language Processing Group*. URL: <https://nlp.stanford.edu/projects/snli/> (visited on 04/06/2019).

- [89] *TensorBoard: TensorFlow's visualization toolkit*. URL: <https://www.tensorflow.org/tensorboard> (visited on 05/15/2019).
- [90] *Share Code. Track Work. Ship Software. | Team Foundation Server - Visual Studio*. URL: <https://visualstudio.microsoft.com/tfs/> (visited on 04/15/2019).
- [91] *InVision | TTTM system Low-fi demo*. URL: <https://invis.io/7WR64K92UXJegin> (visited on 04/11/2019).
- [92] *Redux · A Predictable State Container for JS Apps*. URL: <https://redux.js.org/> (visited on 05/10/2019).
- [93] *Destructuring assignment - JavaScript | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment (visited on 06/01/2019).
- [94] *Promise based HTTP client for the browser and node.js*. URL: <https://github.com/axios/axios> (visited on 04/28/2019).
- [95] *React Documentation*. URL: <https://reactjs.org/docs/getting-started.html> (visited on 05/22/2019).
- [96] *React Router: Declarative Routing for React.js*. URL: <https://reacttraining.com/react-router/> (visited on 05/22/2019).
- [97] Jimmy Ba Diederik P. Kingma. *Adam: A Method for Stochastic Optimization*. Jan. 2017. URL: <https://arxiv.org/abs/1412.6980> (visited on 05/15/2019).
- [98] *Node.js*. URL: <https://nodejs.org/en/> (visited on 05/09/2019).
- [99] *Node.js Passport Documentation*. URL: <http://www.passportjs.org/docs/> (visited on 05/09/2019).
- [100] *Microsoft Azure cloud service*. URL: <https://azure.microsoft.com/en-us/> (visited on 05/22/2019).
- [101] *Amazon Web Service*. URL: <https://aws.amazon.com/> (visited on 05/22/2019).