

Frankfurt University of Applied Sciences



Development of an Automated Integration System for SAPUI5 Web Applications

Master Thesis

Author: Eman Belal
Supervisor: Prof. Dr. Christian Baun
Co-supervisor: Prof. Dr. Thomas Gabel
Submission Date: December 2017

Affidavit

I hereby declare that I have developed and written the enclosed master thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This master thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere. I am aware of the fact that a misstatement may have serious legal consequences.

Eman Belal
December 2017

Acknowledgements

It is with immense gratitude that I acknowledge the support and help of Prof. Dr. Christian Baun for guiding me throughout the thesis and providing me with suggestions and feedback. I would like also to thank Prof. Dr. Thomas Gabel for his support and for being my Co-supervisor for this thesis. Last but not least, I would like also to acknowledge the continuous support of my family which never ends.

Abstract

The use of web applications has been growing over the past years. They are playing an important role in the communication between different clients and servers. While there are new software versions being frequently released, upgrading software applications from one version to another can be challenging. In other words, it may lead to compatibility issues causing system failures. That's why software maintenance is important and should be taken into consideration during the development process. JavaScript is a programming language that is well suited for the development of web applications. Due to its dynamic features, it becomes a very challenging problem in the analysis of web applications.

This paper discusses the methods implemented for developing an automated integration system for the JavaScript framework SAPUI5. This solution is intended to help users, who want to upgrade their SAPUI5 applications, by performing an upgrade check to their existing applications for detecting compatibility issues. This is mainly accomplished by implementing a module for the execution of the upgrade check, in addition to a core parser for analyzing and parsing the JavaScript source code of the SAPUI5 application. Besides, the solution also involved the implementation of another module, which is responsible for indexing and maintaining the SAPUI5 library by defining a proper model of the library with the important information required for the checking the upgrade compatibility.

List of Figures

1.1	Example of a prototype-based object system	3
2.1	SAPUI5 architecture [11]	12
2.2	Model View Controller [13]	14
2.3	SAPUI5 folder structure [19]	15
2.4	SAP HANA DB Architecture [10]	19
2.5	Screen-shot of the Jenkins Job Configuration [3]	22
2.6	Git shared repository	23
2.7	Git staging area [2]	24
3.1	The folder structure of the SAPUI5 Library in Nexus Sonatype repository .	28
3.2	ERD of the SAPUI5 Library	29
3.3	Architecture	31
3.4	Flow chart of the Solution	34
4.1	The project structure	36
4.2	Macroscopic View of the UML Class Diagram	37
4.3	Example showing the configuration of the “bower.json” file	38
4.4	Executing bower install example	38
4.5	Package created on HANA XS	39

4.6	SAPUI5Lib table on HANA	40
4.7	Microscopic view of Lib Class Diagram	41
4.8	SAPUI5 imported controls within the JavaScript file	44
4.9	Microscopic view of the Class Diagram of “File.node”	45
4.10	Microscopic view of the class diagram of “ModelDefinition.node”	46
4.11	Microscopic view of the UML class diagram of UpgradeHelper.api.node module	47
4.12	Microscopic view of the UML class diagram of builder.node module	48
4.13	Microscopic view of the UML class diagram of UpgradeHelper.parser.node module	49
4.14	manifest.json file containing the SAPUI5 version	50
4.15	OData service defined within the “manifest.json” file	51
4.16	Microscopic view of “Parser.js” class diagram	57
4.17	Microscopic view of “UpgradeHelper.db.node” class diagram	59
4.18	Microscopic view of “DocuGenerator.api.node” class diagram	60
4.19	Microscopic view of “DocuGenerator.parser.node” class diagram	61
4.20	Installing the npm package on the Jenkins job	62
4.21	The user interface of Jenkins Build with Parameters	63
4.22	Configuring the Git repository URL under Source Code Management	63
4.23	Poll SCM configured under Build triggers section	64
4.24	Jenkins Build Step	64
4.25	Archiving artifacts as a Post-build action	65
4.26	Jenkins archived artifacts	65
5.1	Upgrade Impact Analysis example	67
6.1	Configuring the remote triggering of Jenkins	72

6.2	Protoype example of a UI of the SAPUI5 application	72
A.1	package.json file with the used npm modules	75
B.1	Cover page of the document	76
B.2	Table of contents	77
B.3	General information about the SAPUI5 app	77
B.4	Upgrade Impact Analysis list	78
B.5	Continue: Upgrade Impact Analysis list	79
B.6	List of deprecated controls	80
B.7	List of deprecated properties	81
B.8	Analysis of the used controls within the app	82
B.9	Continue: Analysis of the used controls within the app	83
B.10	Continue: Analysis of the used controls within the app	84

List of Tables

2.1	Static vs Dynamic code analysis techniques [27] [22]	7
2.2	Comparison between Event-driven and Multithreading programming [39] [26]	10

Acronyms

HTML	Hypertext Markup Language.....	1
CSS	Cascading Style Sheets.....	1
DOM	Document Object Modelling.....	2
IDE	Integrated Development Environment.....	5
npm	Node Package Manager.....	9
UI	User Interface.....	11
MVC	Model-View-Controller.....	12
JSON	JavaScript Object Notation.....	13
XML	Extensible Markup Language.....	13
BSP	Business Server Page.....	17
DBMS	Database Management System.....	17
SQL	Structured Query Language.....	17
DB	Database.....	18
MVCC	Multi-Version Concurrency Control.....	18
CPU	Central Processing Unit.....	18
MDX	Multi-Dimensional Expression.....	19
JRE	Java Run-time Environment.....	20
CI	Continuous Integration.....	20
VCS	Version Control System.....	23
SCM	Source Code Manager.....	23
App	Application.....	26
ERD	Entity Relationship Diagram.....	29
KPI	Key Performance Indicator.....	56

Contents

Acknowledgements	iii
1 Introduction	1
1.1 Problem Description	1
1.2 Motivation	2
1.3 Goal of the Thesis	4
1.4 Overview	4
2 State of the Art	5
2.1 Existing solutions	5
2.1.1 Static Code Analysis	5
2.1.2 Dynamic Code Analysis	6
2.1.3 Automatic Computer Upgrading	7
2.1.4 Drawbacks of the existing solutions	9
2.2 Technologies used	9
2.2.1 Node.js	9
2.2.2 SAPUI5	11
2.2.3 SAP HANA	17
2.2.4 Jenkins	20
2.2.5 Nexus Sonatype Repository	22
2.2.6 Git	23

3	Design	26
3.1	Requirements Analysis	26
3.1.1	Existing SAPUI5 Source Code	27
3.1.2	Target Upgrade Version	27
3.1.3	Parsing and Analyzing the existing Source Code	27
3.1.4	Indexing and Maintaining the SAPUI5 Library	28
3.1.5	Comparison between both library versions	30
3.1.6	Results output	31
3.2	Solution Architecture	31
3.3	Use Cases	32
3.4	Solution Workflow	33
4	Implementation	35
4.1	Programming Languages and tools used	35
4.1.1	Programming Languages	35
4.1.2	Tools used	35
4.2	Backend	36
4.2.1	Project Structure & Class Diagram	36
4.2.2	Indexing & Maintaining SAPUI5 Library	38
4.2.3	Indexing SAPUI5 App Source Code	46
4.2.4	Start.js	46
4.2.5	Upgrade Helper	47
4.2.6	Documentation Generator	60
4.3	Jenkins Job Configuration	61
4.3.1	Packaging & installing the Node.js project on the server	61
4.3.2	Configuring a Jenkins job	62

5	Evaluation & Testing	66
5.1	Discussion of the output results	66
5.2	Testing	68
6	Conclusion	69
6.1	Summary	69
6.2	Optimization potentials	70
6.3	Future Work	71
6.3.1	Extending SAPUI5 Source Code Indexing Methods	71
6.3.2	Further Modules & Use Cases	71
6.3.3	Parsing additional file types	71
6.3.4	Remote triggering of Jenkins via a SAPUI5 application	72
6.3.5	Automatic indexing of the SAPUI5 library	73
	Appendix	74
A	npm Modules used	75
B	Upgrade Helper Output Document	76
	References	88

Chapter 1

Introduction

Nowadays, web applications are widely used and have been through a very quick growth over the past years. They are used for the communication between clients and service providers via connecting channels [32]. In effect among others, native deployed applications get more and more substituted by modern web applications.

A web application consists of two sides: a server side, in which the implementation of the business logic and data manipulation are done, and a client side that is responsible for the user interaction via the web browser [40].

1.1 Problem Description

JavaScript is a programming language, which is broadly used in the development of web applications. It is considered to be an object-oriented programming language, which can also be used in expanding and enhancing existing web applications. However, the programming structure of JavaScript is different compared to other object-oriented programming languages such as: Java, C#,...etc. This is because, JavaScript does not support classes as well as encapsulation and well-structured programming. Nevertheless, JavaScript is identified to be more flexible and dynamic [36]. Besides, JavaScript is commonly used during the development of web applications with other languages such as Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) [30].

As a result, all web browsers are now supporting JavaScript as a native browser scripting language. This means that all browser programs are currently written either straightforward in JavaScript or indirectly in different languages that is converted afterwards by the compiler into JavaScript [24].

Software maintenance is one of the key aspects in the development of software applications. This is due to the frequent releases of software updates and new versions by software

developers. The purpose of releasing these updates can be for enhancing the performance, fixing bugs, and providing more features. Therefore, software applications as well as the used frameworks (dependencies) need to be upgraded. As a result, very severe problems might be occurred while trying to upgrade the software application to a newer version [31].

1.2 Motivation

Upgrading software applications is considered to be a major problem due to its big impact. This is because most of the applications now are using shared libraries with common code and thus, upgrading an application to a newer version can lead to various compatibility issues causing application failure [25]. Consequently, customers try to avoid upgrading their applications in order to prevent any chances of software failure [25]. However, they are forced to upgrade their applications, when the maintenance support period is over.

SAPUI5 is an example of a JavaScript framework, which is available in different versions according to each release. Therefore on upgrading SAPUI5 version to a later one, developers as well as end users must ensure that there are no compatibility issues that can result in application break or failure [13].

Static analysis of web applications are considered to be a challenging problem when it comes to the use of the JavaScript language. This is due to the following characteristics of JavaScript, which makes it prone to developer errors and mistakes: [36]

- **Dynamic:** which is done by using the `eval` function. Consequently, methods and fields can be accessible which can allow the contents of the objects to be easily altered by means of code injection [36].
- **Interpreted:** This is due to the lack of the compiler availability. Therefore, errors and wrong code can be detected at run-time without prior knowledge by the developers [30].
- **Complex event-based Technique** This technique is used in the communication between JavaScript and the Document Object Modelling (DOM) while changing the DOM tree during run-time [30].
- **Weakly typed:** Which means that data types are not declared. In addition to that, the source code is only checked at run-time by checking the accessibility of the calls and fields [36].
- **Prototype-based:** Unlike other programming languages, JavaScript makes use of a prototype-based object system which is considered to be a class-free technique [30]. Each object contains a collection of different properties, each is assigned

to a value [36]. Moreover, inheritance is supported between objects, which means that properties can be straightforwardly inherited from other objects [30]. In other words the current object and its parent are investigated in a loop, in order to search for a specific property.

The process of restricting how an object can behave is very tough to achieve, due to the flexibility of the object system of the JavaScript language. This means that any object can be easily altered by changing the content of its prototype field and thus leading to a great impact on all the other referenced objects. The code in Figure 1.1 is an example showing that whenever the function `foo()` is called, a property `prop` will be altered for the object `obj` [36][30].

```
function foo(obj, prop, value){  
    obj.prop = value;  
}
```

Figure 1.1: Example of a prototype-based object system

As a result, researches have been made to control the dynamic behavior of the JavaScript language. This is accomplished by using some static analysis and compiler techniques for parsing and analyzing the JavaScript code, in order to enhance the performance, security and correctness of the web applications.

1.3 Goal of the Thesis

The goal of this thesis is to develop an automated integration system for SAPUI5 web applications. This involves the implementation of a core parser tool, in order to analyze the JavaScript source code of the SAPUI5 applications. Furthermore, this parser can be a powerful base for additional modules like an upgrade helper, which notifies users about the risk on upgrading their applications in addition to detecting the compatibility issues between both versions. The solution can also support the generation of a technical documentation about the given application.

In this thesis, the focus is on the Upgrade Helper module and how it is used for checking the upgrade compatibility of SAPUI5 applications. Moreover, the thesis also gives an overview example of reusing the core parser by implementing another module for generating a technical documentation of SAPUI5 applications.

1.4 Overview

In Chapter 2, a description of the state of the art technologies and methods will be briefly discussed. Chapter 3 explains the design of the solution and the needed requirements, as well as the architecture of this solution. The implementation part will be discussed in Chapter 4. In Chapter 5, a brief evaluation of the output result and how the developed solution was tested will be discussed. In Chapter 6, the conclusion of the overall performance of the solution will be stated, in addition to what is required to be integrated to the tool for future enhancement.

Chapter 2

State of the Art

In this chapter, the state of the art methods and technologies are discussed. In the first section, some of the existing solutions used for analyzing web applications, as well as checking the upgrade impact are listed and briefly explained. In the second section, the background information regarding the tools and the technologies used in this thesis are demonstrated.

2.1 Existing solutions

There exist various solutions, which are used for analyzing and checking the quality of JavaScript applications. In addition, some other existing solutions are being used in checking the impact on upgrading the resources of the computer systems. In the following lines, an overview about the current existing solutions will be explained, in addition to the advantages and drawbacks of using them.

2.1.1 Static Code Analysis

Static analysis is a technique which is used for assessing the program code without the need to execute it. Compared to manual code review or writing test cases, static analysis tools are considered to be more powerful and less time consuming. Particularly, they are aiming for checking the fulfillment of the code quality, by catching problems that are related to code or programming best-practice guidelines. Static analysis tools are usually integrated within the Integrated Development Environment (IDE) to show errors during development.

These tools can be used for example, to determine the locations in the code which result in an array overflow or a null pointer. In addition to that, static analysis tools are used

to detect wrong code behavior that won't lead to exceptions or errors, for instance those impossible comparisons between incomparable objects. Some static analysis tools are also used in detecting issues resulting from code styling; those which are related to naming conventions or using the curly braces inside if-conditions or loops [21].

There exist several examples of JavaScript static analysis tools, called linting tools, such as the following:

- **JSLint:** JSLint is considered a static analysis tool, which is developed in JavaScript. It is used for checking and validating JavaScript syntax coding. Basically, it analyzes the JavaScript or JSON source code, which is provided as an input to the tool. In other words, it searches for any syntax error in addition to any problems related to the structure of the code and naming conventions. If a problem is detected, the JSLint returns a message to the user explaining each problem as well as the location where it is occurred [5].
- **JSHint:** JSHint is also a static analysis linting tool, which ensures code quality by checking and validating JavaScript code. One of the advantages of using JSHint over JSLint, is that the configuration can be customized by adding our own set of rules to the configuration file [4].
- **ESLint:** ESLint is also another example of static analysis code linting tools. It is developed in Node.js. As a result, it can be installed using npm, and thus allowing a rapid run-time environment. It also provides custom configuration by adding new rules. ESLint is also characterized by being pluggable, which means that each rule can be easily specified at run-time [1].

2.1.2 Dynamic Code Analysis

Dynamic code analysis is a technique which is used to examine the source code while executing it. In other words, the program has to be running in order to perform dynamic analysis to its code. Code testing and profiling are considered examples of dynamic code analysis. During the analysis of the running program's code, one or more execution paths are covered [27]. However, It is characterized by the following features [22]:

- **Input dependent:** This means that the behavior of the program is affected by the given inputs.
- **Precise:** Since one or more execution paths are examined and not the whole program. This results in a better precision of the extracted information during the analysis.

Moreover, the execution of dynamic analysis is accomplished by having a collection of defined test cases, thus forming a big test suite [27]. On the other hand, one of the disadvantage of using dynamic code analysis technique is the lack of providing a generalized result for the whole program, but only specific for the current execution path. On the contrary, static analysis techniques provide a full generalized result for the whole program [23].

An overview comparison between both static and dynamic code analysis are highlighted with the following table 2.1:

Table 2.1: Static vs Dynamic code analysis techniques [27][22]

	Static Analysis	Dynamic Analysis
Operation	Without running the program (called “program-centric”)	Program should be running (called “input-centric”)
Program execution paths	All execution paths get examined	One or more execution paths get examined
Domain	Fetching the dependencies of properties is limited due to the various of number of execution paths	Dependencies of the properties can be easily fetched due to the availability of few execution paths
Precision	Low precision	High precision
Usage	Used for ensuring program correctness and code quality	Used for detecting errors and bugs
Speed	Faster	Slower

2.1.3 Automatic Computer Upgrading

Automatic Computer Upgrading is considered to be an invention, which helps users to upgrade their current computer resources’ version to a later one. Since maintainability of the utilities of system resources is one of the important aspects for a computer system, the costs and time must be taken into consideration. The resources of a computer system are being updated very often. Consequently, end users have to examine the availability of upgrades in addition to their impacts on their current systems. This is achieved by

first checking whether an upgrade is available or not, by comparing the version number of each system resource to its upgrade one.

When an upgrade is found, the version of the current system resource in addition to its upgrade version are investigated to get the distinctions between them. Therefore, the end users can be aware of the impact of these upgrades on their current systems.

Knowing the impact on upgrading the resources of the current computer system is achieved by passing through the following phases: [38]

1. Storing the information about the newer version, including features which are related to one or more earlier versions
2. Storing the current version information
3. **Comparison phase:** In this phase, the information of the newer version is compared to the current one
4. **Output phase:** according to the comparison results, a report containing the upgrade information is displayed to the user, which tells whether an upgrade should be done or not. Furthermore, these information can be saved on a portable medium like a CD-ROM or on a database.

The upgrade information can contain information about features of the new version, what is changed compared to the previous version and the reasons for this upgrade.

Advantages:

- Automatic detection of existing upgrades
- Specifying how important is each upgrade
- Knowledge about the relations between the upgrades and their dependencies is provided.
- The ability of the user to decide whether to upgrade each resource or not
- Automatic upgrade of the computer's resources

2.1.4 Drawbacks of the existing solutions

There are some limitations of the previously mentioned existing solutions as follows:

- The static analysis linting tools are only used for checking the quality of JavaScript code
- Limited set of rules during static analysis
- No information about the given application is retrieved
- The automatic computer upgrading invention is limited only for upgrading and checking the impact of upgrading the computer system's resources.
- No user friendly interface in the automatic computer upgrading invention

All these drawbacks are considered powerful reasons towards developing a solution, that is used for helping SAPUI5 users know the risk impact on upgrading their applications. Through an interactive user friendly interface, the source code can be provided as an input and gets parsed.

2.2 Technologies used

This section gives an overview about the technologies used for this thesis.

2.2.1 Node.js

Node.js is defined as a run-time environment for server-side JavaScript. It is using the Google Chrome's V8 as a base engine. It also has a default package manager, Node Package Manager (npm), which is considered to be the biggest public repository of open source libraries worldwide. This is due to the availability of million of free packages that are hosted on the npm registry. In addition to that, npm is very helpful in sharing and spreading code among different people as well as managing the project dependencies. One big advantage of Node.js, is the usage of an event-driven, non blocking I/O model for supporting parallel executions. As a result, it becomes more effective and easier compared to using multithreading techniques [7][8].

Event-driven Model over Multithreading

Event-driven model is based on the notification of registered events. In other words, the application gets an alert, from the notification system, for processing the event whenever it occurs. In addition to that, event-driven programming requires asynchronous I/O, which is very crucial in order to avoid the waiting of application in an I/O operation, and thus prevents being blocked.

On the other hand, Multithreading is a programming approach, in which an application is divided into multiple concurrent processes, each is executed on one thread. This can be problematic while using single-core systems, due to the blocking of threads by the processor till the current thread execution is finished

The following table 2.2 summarizes the main differences between event-driven and multithreading models:

Table 2.2: Comparison between Event-driven and Multithreading programming [39][26]

	Event-driven	Multithreading
Number of threads	Single thread for all requests	Multiple threads, each is used to execute an application request
Environment	Asynchronous I/O environment with callbacks	Multithreaded environment
Blocking	Non-blocking due to Asynchronous I/O execution	Threads are blocked waiting for the execution of the current thread to be finished, which can lead to Deadlock
Memory Allocation	Less memory, since only a callback function pointer and its arguments are stored and not a whole thread stack	A whole thread stack is stored

Programming with Node.js

As previously mentioned, Node.js uses an event-driven, I/O non blocking model for parallel executions. In other words, the Node.js server uses only a single thread for handling all

incoming requests. Non-blocking I/O operations are accomplished in Node.js by having an event loop. Unlike other programming languages, the event loop is constructed during runtime instead of being available as library [7]. The event loop gets initialized directly after starting Node.js and executing the input script. Afterwards, the event loop is started and the node exits it when there are no longer callbacks available. Because of this, JavaScript is chosen to be used in Node.js programming due to its ability for registering event handlers as well as supporting event callbacks [39].

The event loop consists of the following phases [7]:

- **timers:** this phase is responsible for running callbacks of the functions `setTimeout()` and `setInterval()`
- **I/O callbacks:** In this phase, all callbacks are executed apart from close callbacks, those executed by the timers phase and `setImmediate()`.
- **idle, prepare:** This phase is executed internally.
- **poll:** In this phase, any new I/O events will be fetched
- **check:** In this phase, the callbacks of the function `setImmediate()` are executed
- **close callbacks:** This phase is responsible for the execution of the callbacks of close functions, e.g. `socket.on('close', ...)`

2.2.2 SAPUI5

SAPUI5 is a User Interface (UI) technology framework developed by SAP. It is based on JavaScript, HTML5 and CSS [12]. SAPUI5 is used in the development of cross-platform web applications running on all desktops and mobile devices' browsers [15].

Furthermore, various number of standard and extended UI controls are supported within its run-time of client-side HTML5 rendering library. Due to its compliance with OpenAjax, standard JavaScript libraries can also be used in SAPUI5 applications. It is also built on the open source jQuery library as its foundation [12]. In addition, SAPUI5 offers an automatic adaptability of its UI to fit different devices and screen sizes [11]. The following figure 2.1 gives an overview representation about the architecture of SAPUI5.

SAPUI5 Architecture

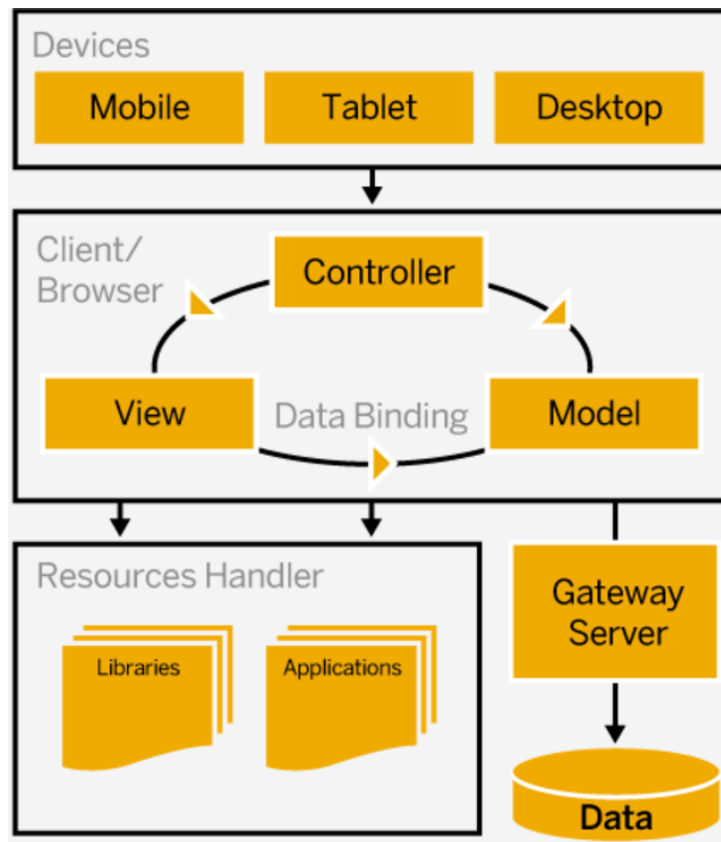


Figure 2.1: SAPUI5 architecture [11]

As shown in the above figure 2.1 and discussed earlier, SAPUI5 applications are cross-platform, which means they can run on any device such as: Mobile, Tablet or Desktop PC. All the SAPUI5 resources and libraries are stored either on SAP Cloud Platform or on an SAP NetWeaver application server according to the used development environment. Simply, a request is sent from the browser to the specific server and thus all related SAPUI5 libraries and resources needed to load the application on the browser are retrieved. Also the model gets initiated and as a result, the business data gets extracted from the database.

SAPUI5 applications are also not allowed to call applications and services from outside its network or domain; therefore, SAP Gateway is needed to allow the external communications and fetching of data by using OData model. As seen also in this figure, SAPUI5 is following the Model-View-Controller (MVC) architectural design pattern which will be discussed in the next page [15].

Model-View-Controller (MVC)

MVC is considered one of the architectural design patterns used in the development of modern interactive web applications. In addition, it is used to isolate the presentation of an application from the underlying logic of data [34]. As the name suggests, it consists of three interconnected components as follows:

1. **Model:** An application's model is considered the core structure implementation of the application [33]. It is responsible for carrying out the data of the application as well as fetching and manipulating data from the database [15].

SAPUI5 supports the use of both client-side and server-side models. Client-side models means that data availability, manipulation and filtering are implemented on the client side without the use of the server. In addition, client-side models support only the use of small data sets.

On the other hand, server-side models are those models in which data is available on the server-side and can only be retrieved using server requests. As a result, server-side models are intended for large datasets.

The following are the list of predefined models, which are supported by SAPUI5 [13]:

- JavaScript Object Notation (JSON) model: It is considered a client-side model, in which data-binding is applied to a small dataset of JavaScript object data. In addition, two-way data-binding is supported by the JSON Model.
 - Extensible Markup Language (XML) model: It is a client-side model, in which data-binding is applied to a small dataset of XML. In addition, two-way data-binding is supported by the XML Model.
 - Resource model: This model is used for data that are related to resource packages. Particularly, it is responsible expressing data in various number of languages. This model supports only one-time binding.
 - OData model: This model is considered a server-side model, in which data is available on the server side. The databinding is applied to data that is retrieved using OData services. It supports also two-way data-binding.
2. **View:** The view is responsible for the output UI representation of an application [15]. In other words by applying data binding, data is fetched from the model and gets rendered and presented to the user by the view. Besides holding the graphical representation of data, the view can also hold another subview, and thus supporting view inheritance [33].

SAPUI5 supports the use of the predefined views: JavaScript, XML, HTML and JSON views [13].

3. **Controller:** The controller is responsible for handling the user's interactions and inputs. In addition to that, it is responsible for modifying the contents of the model according to user interactions. Consequently, the application's view gets updated with the new model changes [34].

The following figure 2.2 illustrates the architecture of the MVC design pattern:

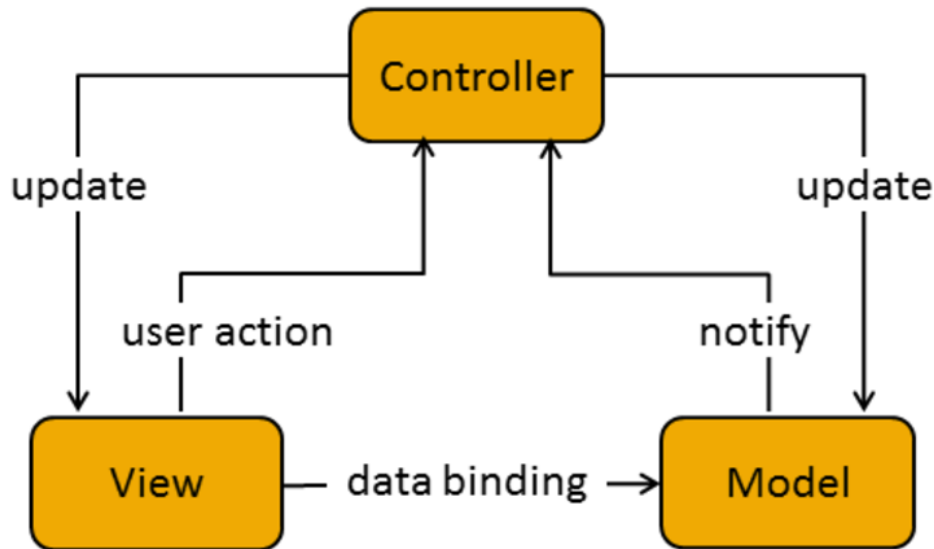


Figure 2.2: Model View Controller [13]

Furthermore, MVC is characterized by being partition-independent. This is due to the existence and execution of the model, view and controller in a single-address space [34].

Using MVC during the development of web applications results in the following advantages [15]:

- Helps in facilitating the code maintainability, readability and extensibility of the application
- Provides the ability to do view modification without affecting the implicit data business logic
- Provides the capability of sharing the same data among different number of views

SAPUI5 Folder Structure

A typical SAPUI5 application, as shown in the figure 2.3 below, consists of a root folder which contains the “webapp” folder. In addition, some sub-folders are located within

the “webapp” folder, which include the views, controllers used in the application. Some additional folders can also be included, which contain additional files such as localization properties, models, odata services and test files [19]. In addition, three important files should also be included within the “webapp” folder, they are explained as follows:

1. **Component.js**: This file is responsible for setting up the application, by defining the runtime metadata in addition to the methods of the components [17].
2. **index.html**: This file is responsible for instantiating and starting the SAPUI5 web application.
3. **manifest.json**: This is the application descriptor file of the SAPUI5 application. It is responsible for defining the application’s metadata, as well as its components and the path to SAPUI5 runtime resources . For instance, the information about the SAPUI5 version used, is specified in the “manifest.json” file.

neo-app.json: This file is included within the root folder of the SAPUI5 application. Likewise the “manifest.json” file, it is an application descriptor file which is responsible for holding all the settings and metadata information for the SAP web development tool “SAP WebIDE”. The SAPUI5 version used by the application can also be specified inside this descriptor file [18]. In addition, configurations related to the path of the SAPUI5 resources are located can be also specified.

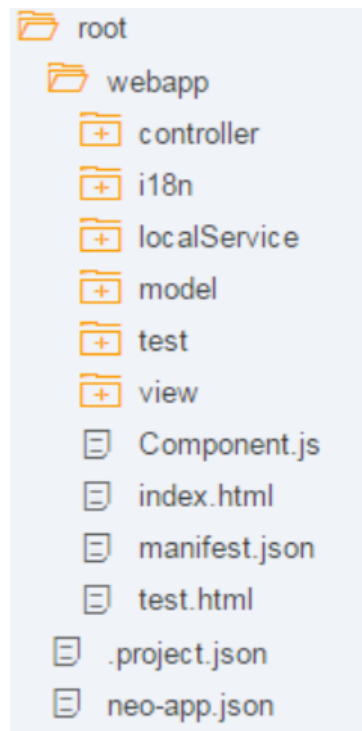


Figure 2.3: SAPUI5 folder structure [19]

Data Binding

SAPUI5 supports the use of data binding operation, which is basically for connecting the UI controls with the relative datasets. Consequently, this ensures that both the application data and the UI control are always synchronized.

There exists three types of bindings as follows [13]:

1. One-way binding: It occurs whenever the bounded UI control is automatically updated, due to changes in the application data model.
2. Two-way binding: It occurs when also any user change on the UI control automatically affects the application data model. This is the default binding mode for SAPUI5.
3. One-time binding: It occurs only once from the data model to the UI control.

SAPUI5 Versioning

Like any other frameworks, there are different versions of SAPUI5 library. As discussed in the previous chapter, each version released is considered an enhancement of the previous one in terms of performance, new features and bug fixes.

SAPUI5 follows a formal convention of using a three-digit version system. In other words, each version is divided into three digits, for example: 1.28.9. Particularly, those digits have different significance behind them, which are illustrated as follows:

1. The first digit is used for determining the major version number.
2. The second digit is used for determining the minor version number.
3. The third digit is used for determining the patch version number.

In other words, both the major and minor version numbers are accompanied by each release. However, the use for the patch version number is just for distinguishing between the patches within the release. This means that no new features are introduced within each patch number. However, the patch is only for modifying and fixing the existing implementation of the major and minor version. [13].

SAPUI5 Control Libraries

There exists various number of UI control libraries that are supported by SAPUI5. For instance, “sap.ui.core” and “sap.ui.layout” are considered to be the base libraries in which any other UI library uses [13].

Moreover, the following lists the main UI control libraries with their usages [13]:

- **sap.m:** It is considered the most important UI control library. This is due to the availability of different responsive UI controls, which are optimized for mobile devices as well as desktop [15]
- **sap.ui.commons:** This library provides a large number of standard UI controls. Currently, this library is deprecated since version 1.38 and “sap.m” is used instead [14].
- **sap.makitt:** This library provides the makitt chart controls. This library is also deprecated since version 1.38, and “sap.viz” control library is used instead [14].
- **sap.viz:** This library offers the use of chart controls. These controls are related to the VIZ charting library.

The SAPUI5 ABAP Repository

The SAPUI5 ABAP repository is a Business Server Page (BSP) based repository, in which SAP customers can use for storing their SAPUI5 applications and components. It is considered a component of the SAPUI5 ABAP back-end infrastructure. The idea of using this repository is to create a BSP application repository for each SAPUI5 application stored [16].

2.2.3 SAP HANA

SAP HANA is an in-memory relational Database Management System (DBMS) [10]. It is considered the center of the SAP HANA Appliance. Using SAP HANA, all processes that are related to business and analytic-specific logic are handled, in addition to other persistent operations.

Due to the high demand of applications support, SAP HANA plays an important role in supporting the recent business applications. This is accomplished by handling the services related to data management in effective ways compared to other Structured Query Language (SQL) based data management systems which is restricted to handle these

requirements [28]. For instance, logical procedures required for analytical applications such as: clustering and analysis are easily achieved using SAP HANA unlike SQL which can not perform these kind of procedures. In other words, to accomplish those kind of operations using SQL the application fetches the data from the Database (DB) to the application and the data gets handled afterwards [29].

Moreover, using SAP HANA various standards of transaction isolation is achieved by having the support of the so-called Multi-Version Concurrency Control (MVCC). In addition to that, the design of SAP HANA DB makes it a powerful connection between the application layer and the data management layer [37].

In-Memory DB

Being an in-memory DB, means that the whole data is being stored in the main memory. This is due to the availability of the various number of multi-core Central Processing Units (CPUs) having huge main memory storage which SAP HANA is executed on. As a result, this helps SAP HANA to overcome the overhead of performing a lot of disk I/O, and thus accelerating the performance of the SAP HANA DB. Nevertheless for ensuring that data is permanently preserved, a backup should be applied on disk drives as an asynchronous background task which does not affect the DB performance [10].

SAP HANA DB Architecture

As mentioned earlier, the whole data is stored in the main memory of the SAP HANA DB compared to other traditional relational databases. As a result, data engines are considered to be the core of the SAP HANA DB in which the data is stored either in row or column-based layout. In addition to that, different kinds of engines can be embedded, due to the ability of the HANA system to be extended. These engines are responsible in storing the data in the main memory, provided that there exists sufficient storage [29].

The following figure 2.4 shows the architecture of the SAP HANA system which illustrates how the data is being stored.

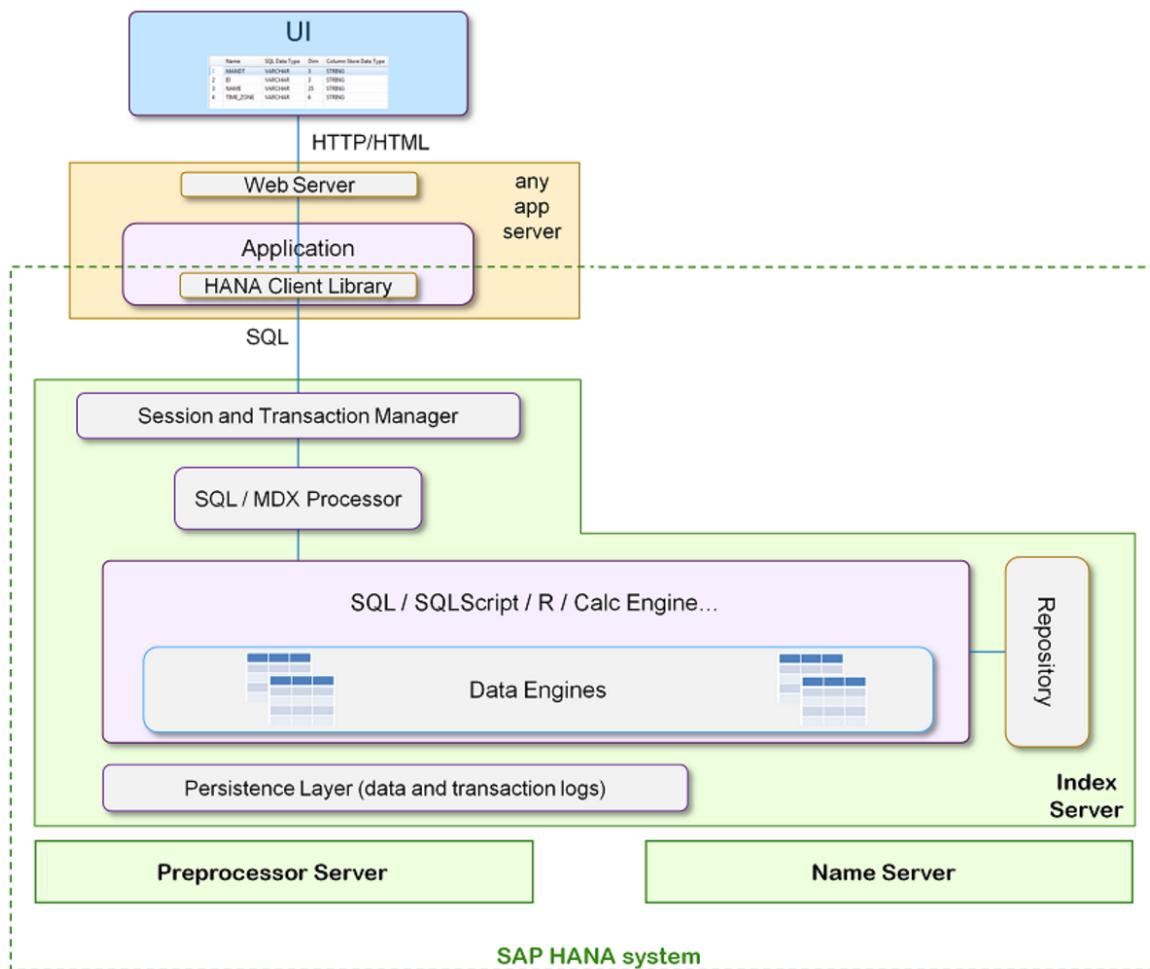


Figure 2.4: SAP HANA DB Architecture [10]

As shown in figure 2.4, three types of servers exist:

- **Index Server:** This server is responsible for the data storage and processing engines, which makes it the most important element of SAP HANA DB. In addition, it is using sessions and transactions that are authenticated for handling SQL or Multi-Dimensional Expression (MDX) statements.
- **Preprocessor Server:** This server is used by the index server. using this server, the text data is examined in order to get all the needed information on which the text search capabilities depend.
- **Name Server:** In this server, all the information about how the components of SAP HANA system are organized is stored. This information includes the locations of data as well as the server on which it runs.

In addition, SAP HANA database consists of a persistence layer which guarantees the latest committed state after system restart and that the execution of all transactions are either completed or not. In other words, it is used to ensure the durability and atomicity of SAP HANA transactions.

The SAP HANA database supports the use of a specific scripting language called SQLScript. Using SQLScript, the whole data complex logic of the application gets inserted into the database. On the other hand, using SQL only insert part of functionality into the database and not the whole data logic compared to SQLScript. SQLScript is considered to be more parallelized using various number of processors.

Besides SQLScript, SQL as well as developing programs using R language are also supported by SAP HANA. In other words, both SQL and SQLScript are based on data engine functions that are built-in. These functions are considered a gateway to multiple metadata definitions that is saved in one mutual catalog ex: tables, columns, procedures and views.

2.2.4 Jenkins

Jenkins is considered an automation server, which is open-source and implemented using the Java programming language. Therefore, the machine should have Java Run-time Environment (JRE) installed in order to install Jenkins on it. Using the Jenkins server, different jobs like: building, testing and software deployment of any project can be automatically executed. In addition, Jenkins is characterized by the following features [3]:

- Provides Continuous Integration (CI) and Delivery
- Can be easily installed and run directly on different operating systems ex: Windows, Mac OS X and other Unix
- Provides straightforward configuration with the help of its simple web interface
- Supports many plugins and thus, provides integration with all the CI and continuous delivery systems.
- Ability to be extended to support different jobs due to the availability of various number of plugins
- Provides fast distribution among various number of platforms and machines

Setting Up Build Jobs with Jenkins

As mentioned earlier, Jenkins can be used for building and testing software projects. By creating a so called “free-style software project” job on Jenkins, some repetitive tasks can be performed and the project can be configured to be built periodically. In order to achieve this, the following sections should be specified [3]:

- **General:** General information about the project ex: project name in addition to the type of this project.
- **Source Code Management:** Location of the source code such as Git or a Subversion.
- **Build Triggers:** This identifies when the build should be triggered by Jenkins ex: periodically, remote build, after some projects are build, ..etc
- **Build Environment:** Information about the build environment settings, for example providing configuration files and/or build with Ant.
- **Build:** These are the actual build scripts in which build is executed. There different Build script types: ant, maven, shell script, batch file, etc.
- **Post-build Actions:** This section defines how the build output should be handled. For instance it can be configured to archive the artifacts and/or documenting the javadoc and test results. In addition, it can be also configured to send a notification email with the build results to anyone.

The following figure 2.5 shows a screen-shot of the Jenkins Job configuration interface:

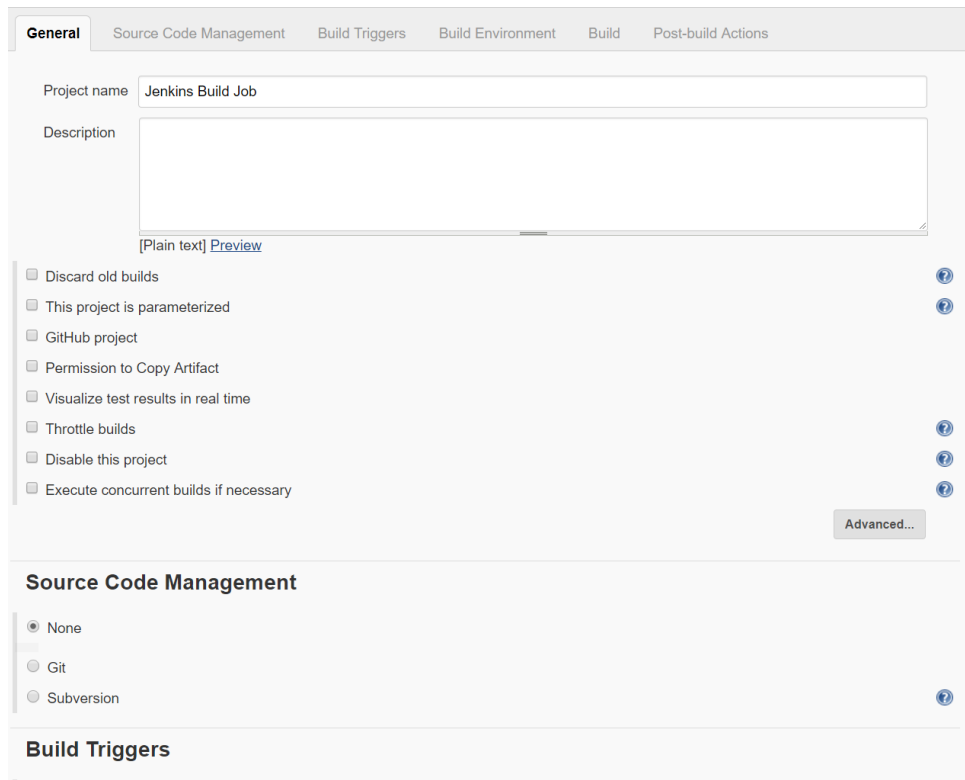


Figure 2.5: Screen-shot of the Jenkins Job Configuration [3]

2.2.5 Nexus Sonatype Repository

During the life cycle of the development process, developers need to have an internal repository which allows components to be distributed and shared among them. As a result, Nexus Sonatype provides a software repository manager, which is used to deal and manage software packages and components. These components can be a library or framework. It can also be a whole application or a static resource such as an image or file. In addition to that, these components are usually stored in a compressed folder that consists of multiple files.

Moreover, Nexus Sonatype repository manager provides the ability to search for any software component. It is also used for handling the metadata of the components, as well offering the facility to show all the component dependencies. In addition, Nexus Sonatype repository helps the user in the integration with other external components [6].

Advantages

The followings are some of the advantages of using the Nexus Sonatype repository manager [6]:

- Helps users to download off the local repository manager in a rapid way, and thus accelerating the build performance of the software
- Supports caching of the components, which results in smaller bandwidth to use.
- Allows the usage of unified techniques for the users to get the components.
- Unified accessibility for accessing the configurations to remote repositories, as well as unified storage of all the components that have been used.

2.2.6 Git

Git is considered a powerful Version Control System (VCS), which is used for managing projects. Being free and open source, makes it available and used by everyone [2]. It is used as a Source Code Manager (SCM) tool, which is used for keeping track of all the information of every version of the project. As a result, it is very useful in the development of big projects which involve the contribution of multiple developers.

In other words, Git contains a repository database, in which all project's relevant information about every version is stored. Compared to other VCSs, a copy of the repository is provided in addition to the complete copy of project's related information and files.

The following figure 2.6 shows a representation of the git shared repository used by different developers:

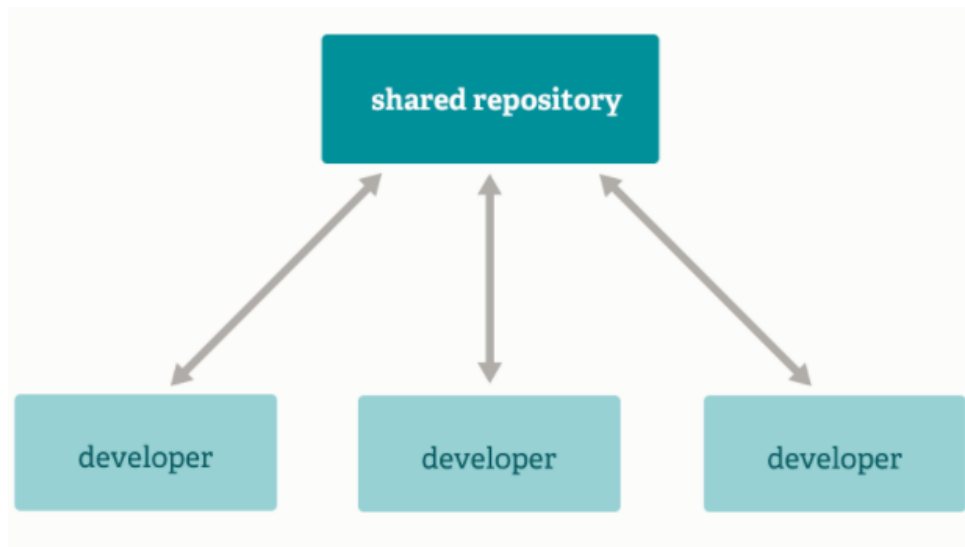


Figure 2.6: Git shared repository

Furthermore, every repository is preserved with values of multiple configuration settings ex: user's name and email address. These configuration settings are specific for each repository, which means that they can not transfer from one repository to another. However, these configurations are investigated according to the site, user and repository [35].

Staging Area

Git supports the so called “staging area” as shown in figure 2.7. This area is considered to be an in-between area before applying the final commit. In this area, the source code to be committed is checked and reviewed. On contrast to other VCSs, Git provides the possibility to stage only a selection of some files and commit them without the need to staging all modified files.

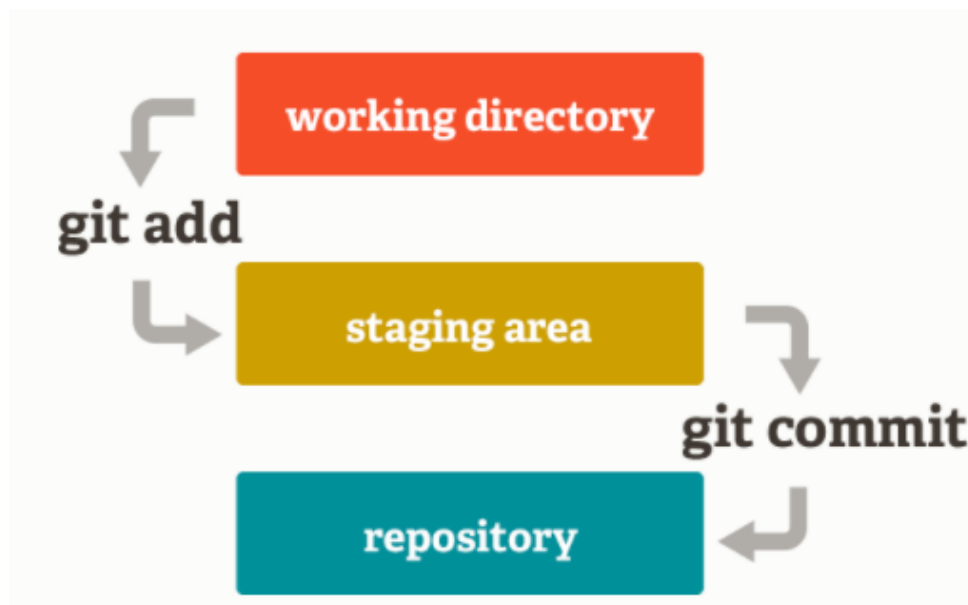


Figure 2.7: Git staging area [2]

As shown in figure 2.7, typing the command “**git add**” on the command line will transfer the modified files into the staging area to be reviewed. Afterwards, these changes are committed to the user's local repository using the command “**git commit**”.

Advantages of Git

There are different advantages which makes Git very powerful compared to other VCS. The following are some of its main advantages [2]:

- **Easy to learn**
- **Quick performance:** This is due to the execution of all the operations locally, and thus avoids the overhead communication with a server.
- **Supports branching and merging:** Multiple branches are supported within Git. These branches can be independent or merged with other local branches.
- **Distributed:** This is due to the possibility of making a “**clone**” to the whole repository and not only doing a “**checkout**” of a piece of the source code.

In other words, the main server is fully backed-up by all the users during a centralized workflow. Consequently, this plays an important role in avoiding any crash or failure since the main server can be substituted by any of the repository backup copies.

In addition to that, Git provides a control mechanism between developers' actions. For example, it prevents a developer from pushing his changes to the repository as long as there exists an unfetched version of the source code.

- **Secure:** This is achieved by making sure that the data is cryptographically protected against any attacks. This is due to the checksum that is applied to each commit, and thus ensuring that data is same on check out.

In addition, each commit is assigned a commit ID which can never be modified unless the ID is changed to a different one.

Chapter 3

Design

As discussed earlier, the goal of this thesis is to implement a solution which can be used to help SAP customers in checking whether they can upgrade their SAPUI5 web applications or this can result in some failure and compatibility issues. This chapter gives an overview about the design of the solution by stating the requirements analysis, the architecture of the solution as well as the different use cases which will be taken into consideration during the implementation.

3.1 Requirements Analysis

The following list shows the different requirements needed for implementing the proposed solution. They are explained as follows:

1. Existing source code of a SAPUI5 Application (App).
2. Target upgrade version.
3. Parsing and analyzing the existing source code.
4. Indexing and maintaining the SAPUI5 library.
5. Comparison between both library versions.
6. Results output.

3.1.1 Existing SAPUI5 Source Code

A SAPUI5 source code should be provided as a user input. The input source code can be provided in various ways; Git repository URL, Zip upload. In addition, SAPUI5 applications can be also accessed from the SAPUI5 ABAP BSP Repository. Currently in this solution, the focus will be on providing the input as a Git repository URL. As a result, the source code can be easily indexed by cloning this repository to the workspace.

3.1.2 Target Upgrade Version

The second important input is the target upgrade version of the SAPUI5 library. As mentioned in the previous chapter, a SAPUI5 version always follows the three-digit version system, for example “1.44.8”.

Besides, the information regarding the current available SAPUI5 library versions can be easily found on the SAPUI5 Development Kit page.

3.1.3 Parsing and Analyzing the existing Source Code

In this step, the source code of the existing SAPUI5 App will be analyzed. Consequently, all the important information required for checking the impact on upgrading the current existing App will be extracted. In other words, the parser will be able to provide the following information to the user:

- General information about the App such as: SAPUI5 version, namespace, component name and OData service used
- Information about the views and fragments, which are used within the App
- Information about the used standard controls within the views
- Information about the used controllers and objects, in addition to the imported standard controls
- Information about the functions used. For instance, the parser will extract the function name, parameters and return value
- The properties used within each standard control

All of these information can be retrieved by having different parsers for supporting the different types of files used within the application. For example, a JavaScript parser is required for parsing a SAPUI5 controller and XML parser is required for parsing SAPUI5 XML views and the same applies for other file types.

3.1.4 Indexing and Maintaining the SAPUI5 Library

The different versions of the SAPUI5 library are available internally on SAP Nexus Sonatype package repository. The capability to extract all the information about each version of the SAPUI5 library is very crucial in the implementation of the proposed solution.

As shown in the below figure 3.1, each version of the SAPUI5 library is typically stored in a big folder containing the whole SAPUI5 resources, documentations and general information.



Figure 3.1: The folder structure of the SAPUI5 Library in Nexus Sonatype repository

As a result, a model definition is highly needed for maintaining the necessary information of each SAPUI5 library such as: SAPUI5 version, UI control libraries, controls, functions, parameters, ...etc. At the end this model can be stored on the SAP HANA DB, which facilitates the accessibility process of the data at any time.

Entity Relationship Diagram (ERD)

The following figure 3.2 shows the ERD representing the model definition of the SAPUI5 library.

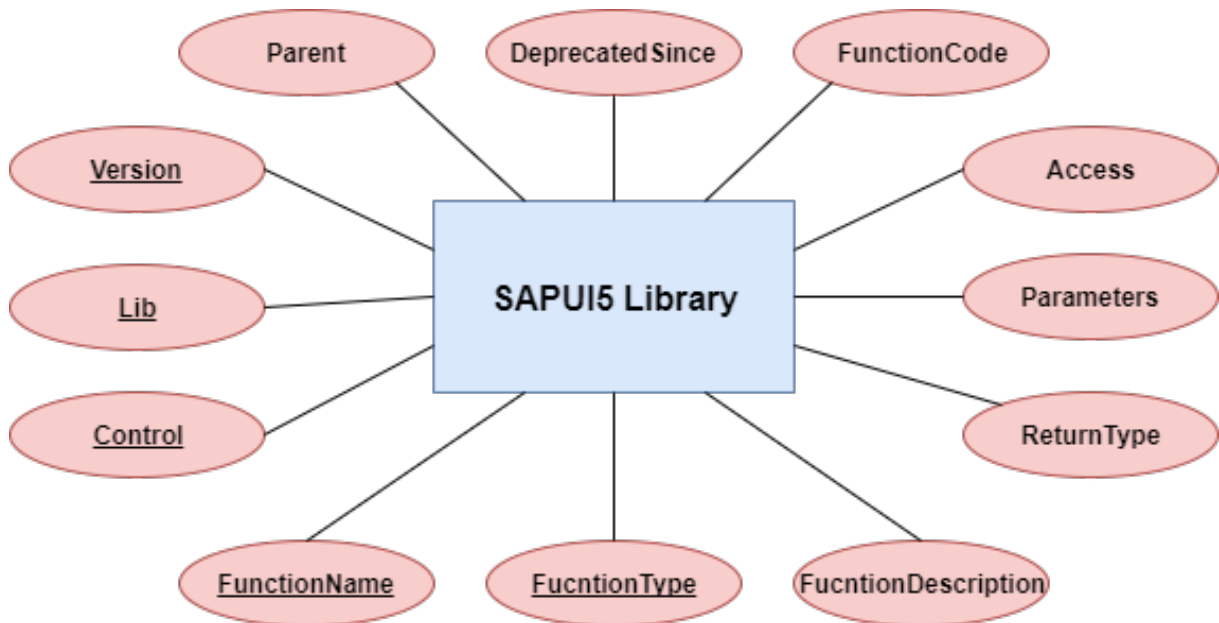


Figure 3.2: ERD of the SAPUI5 Library

The model is defined by having an entity of the SAPUI5 library with the following attributes:

- **Version:** The information regarding the version of the SAPUI5 library is stored. It is expressed using varchar datatype of size 7, due to the using of the 3-digit version convention.
- **Lib:** This attribute holds the information about the SAPUI5 library ex: sap.m, sap.ui.commons,...etc.
- **Control:** This attribute holds the information about the SAPUI5 control.
- **FunctionName:** This attribute holds the name of the function used by this control.
- **FunctionType:** This attribute holds the information of the type of this function, for example: “constructor”, “prototype method” ,“namespace” or “event/property”.
- **FunctionDescription:** The description of this function is stored in this attribute.

- **ReturnType:** This attribute holds the information about the return type of this method, for example: “Boolean”, “Button”,...etc.
- **Parameters:** The parameters of the function are stored.
- **Access:** This attribute holds the information about the type of accessibility of this function. For instance, it can be “public”, “private” or “protected”.
- **FunctionCode:** Here the coding of the control class and the specified function is stored.
- **DeprecatedSince:** This attribute is very influential on checking the impact on performing the upgrade. This is because it determines whether the control or one of its functions is deprecated, in addition to since which version was it deprecated. For instance, if an existing SAPUI5 App with “1.28.9” and using a specific function within the “sap.m.Button” control which is deprecated since “1.30.2”. As a result, upgrading this app to a version which is greater than “1.30.2” would result in a compatibility issue and failure, due to the deprecation of this function in the new SAPUI5 version.
Moreover, some alternative solutions can be provided to the user to use instead of this deprecated standard control or property.
- **Parent:** This attribute is holding the information about which super class (parent) is inherited by this control.

3.1.5 Comparison between both library versions

This is the most important requirement for the development of the proposed solution. The two versions of the SAPUI5 library will be compared against each others. The result of this phase is considered the key indicator of the impact analysis on upgrading the current version of the SAPUI5 App. This is because the comparison will output whether there are changes made in the new version.

This can be achieved by comparing the coding of each control used in the application with its coding in the target version. In other words, the two versions of the controls gets compared line by line, in order to fetch any differences between them, in addition to detecting what changes has been added to the new version as well as what has been removed.

Moreover, the information about the difference between the two versions for each control can be used afterwards for calculating the total amount of changes made to each control. This can be done by comparing the number of changed lines (added + removed) against the total number of lines in the coding of this control in the current version. Finally, all

these information can be used in determining the upgrade risk factor. The risk factor is calculated by multiplying the amount of changes with the number of occurrence, in which this control has been used in the whole application.

3.1.6 Results output

This will be the final output result of the solution. The output will be generated as a word document (.docx) report. This report will contain a summary of the upgrade check made on the requested SAPUI5 App. It will notify the user about the impact on upgrading their SAPUI5 Apps. In addition, it will help users by identifying the issues that arose during the comparison check.

3.2 Solution Architecture

In this section, the architecture of the proposed solution will be explained with the following figure 3.3:

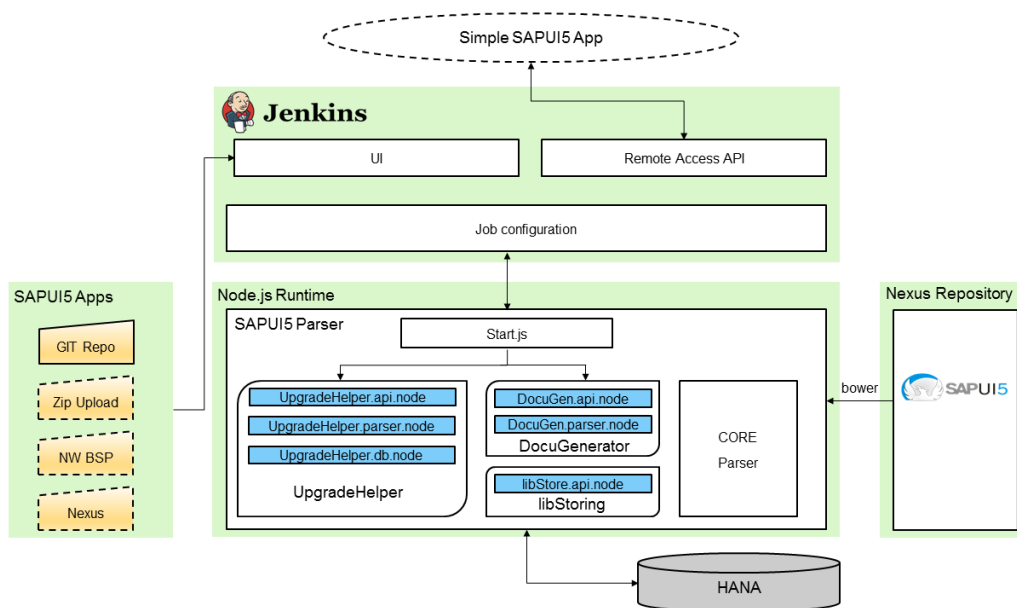


Figure 3.3: Architecture

As shown in figure 3.3 and discussed earlier, the SAPUI5 library will be indexed and downloaded from the Nexus Sonatype software repository using Node.js. This can be achieved with the help of the npm helper module “bower” which will be explained later

in the next chapter. Afterwards, a model definition will be created by extracting all the necessary information of the SAPUI5 library. Eventually, these information will be stored on SAP HANA DB.

The URL to the Git repository of the SAPUI5 source code will be provided as an input via Jenkins. After filling all the information needed as input from the user, a Jenkins parameterized job will be triggered to clone the Git repository to its workspace and the “Start.js” Node.js script gets called afterwards. This Jenkins job will be configured to fetch the user inputs as parameters; the target SAPUI5 version, the URL to the Git repository of the source code in addition to other user inputs.

The back-end part will be implemented using Node.js runtime. The “Start.js” is the first module to be executed within the Node.js project. Accordingly, the two shown modules “UpgradeHelper” and “DocuGenerator” get called. A core parser module will be implemented for performing the static analysis of the given SAPUI5 App. The parser will be adaptable to handle different file types such as: “JavaScript” and “XML”. Moreover, the parser will be used in the creation of the model definition in addition to maintaining the SAPUI5 library on SAP HANA DB.

As also shown in this figure 3.3, a connection between Node.js and HANA database will be established for data storing and accessing purposes. The dotted boxes shown are features which have not been implemented but planned for future work (see section 6.3).

3.3 Use Cases

There are different use cases that will be taken into consideration during the implementation of the proposed solution. They are listed as follows:

- The user has to provide the target upgrade version as well as the Git repository to the source code of the app to be checked for upgrade
- Cloning the Git repository provided by the user
- Extracting the functions, properties and controls used within the app
- Checking the usage of a deprecated standard control
- Checking the usage of a deprecated function/property within a standard control
- How often a control has been used in the App
- Differences between both the current and target SAPUI5 versions with the amount of changes made to the new one

- The impact analysis if an upgrade is performed by calculating a risk factor

The information about deprecated standard controls and functions can be retrieved from the source code. This is due to the availability of deprecated comment flags, which are marked by SAPUI5 developers. In addition to that, some information can be added by the developers like providing an alternative solution, which is not deprecated and yields the same result [13].

3.4 Solution Workflow

As shown in the following figure below 3.4, the workflow of the solution can be divided into the following steps:

1. **Start Jenkins and go to the configured job:** Jenkins is considered as the start point of executing the solution. In this solution the job is called “UpgradeHelper”.
2. **Build With parameters:** By clicking on “Build with Parameters” feature, the required inputs are provided by the user. The inputs are simply two checkboxes for the users to decide which module should be executed; UpgradeHelper and Docu-Generator.
3. **Execution of Node.js script Start.js:** In this step, the Node.js backend gets triggered by executing the “Start.js” module. Afterwards according to the provided user inputs, the respective module gets executed and the output report is generated.

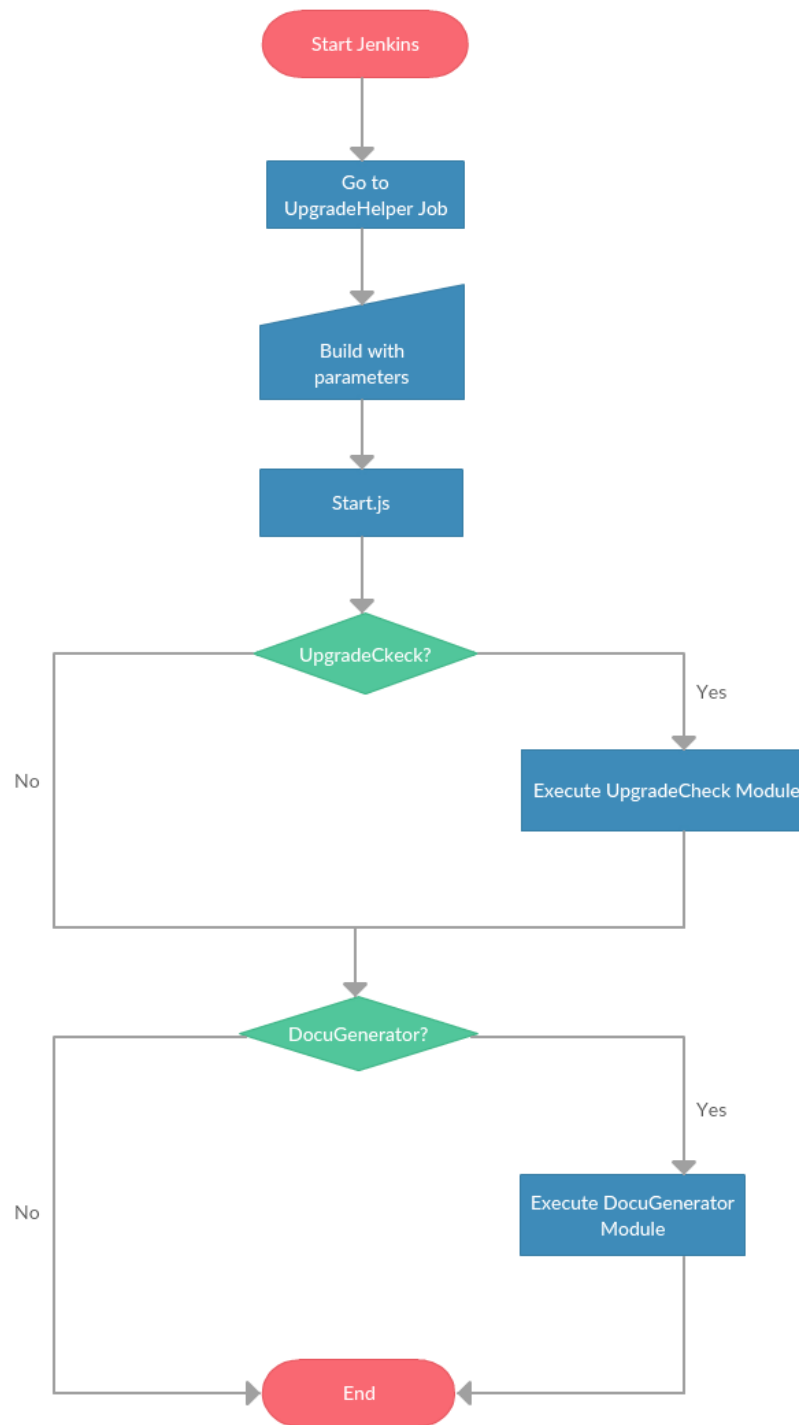


Figure 3.4: Flow chart of the Solution

Chapter 4

Implementation

This chapter gives a detailed explanation on how the solution is implemented. As mentioned in the previous chapter, the implementation of the solution is separated into client side and server side.

4.1 Programming Languages and tools used

This section highlights the programming languages used in the implementation of this solution, in addition to the used tools and development environment.

4.1.1 Programming Languages

- Node.js
- SAP HANA SQLScript
- SAPUI5 (JavaScript, HTML and CSS)

4.1.2 Tools used

- JetBrains WebStorm 2017.1.4
- SAP HANA Studio
- Jenkins

4.2 Backend

In this section, a detailed explanation on the backend implementation of the solution will be discussed. As mentioned earlier, the solution is implemented in Node.js, due to the availability of various number of helper modules on the npm repository (Please refer to appendix A for the complete list of the used npm helper modules).

4.2.1 Project Structure & Class Diagram

The first figure 4.1 shows the structure of the Node.js project. Since it is a Node.js project, the package.json file is used for holding the project metadata and dependencies. The next figure 4.2 is a macroscopic view of the class diagram of the implemented solution, showing all the classes used and the relationships among them.

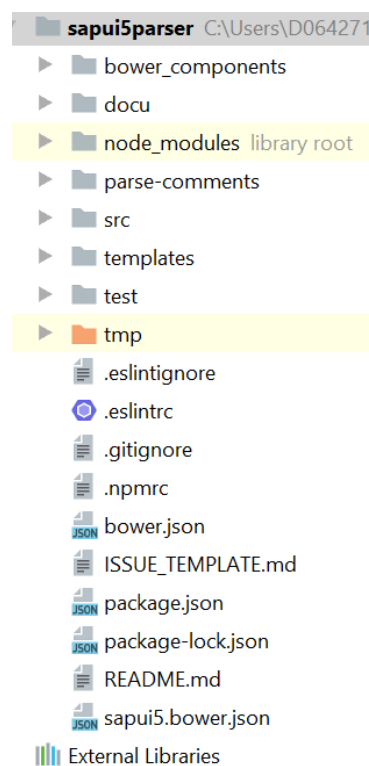


Figure 4.1: The project structure

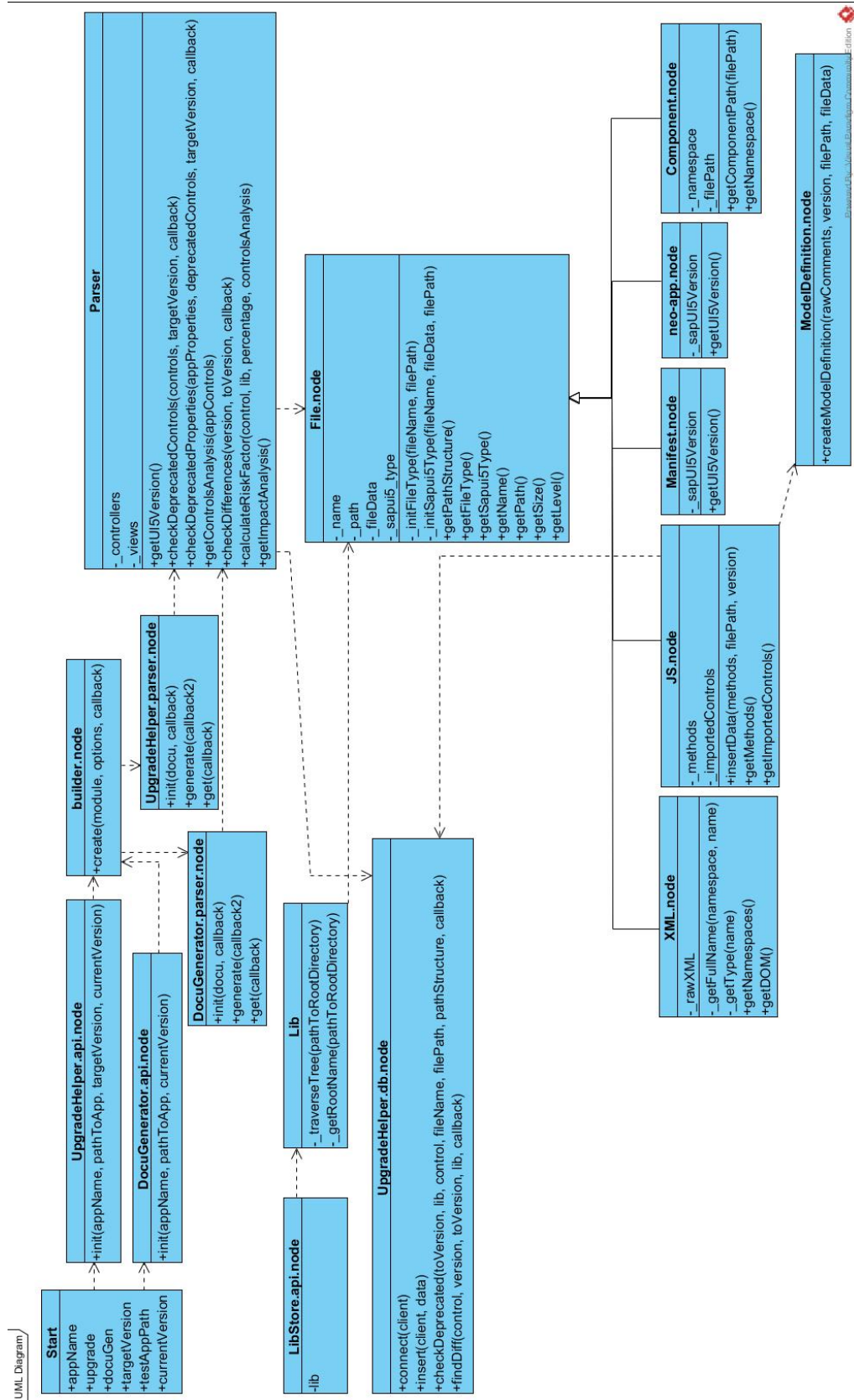


Figure 4.2: Macroscopic View of the UML Class Diagram

4.2.2 Indexing & Maintaining SAPUI5 Library

This section illustrates how the SAPUI5 library is indexed from the Nexus Sonatype repository. This is followed by explanation of the next step which is maintaining the SAPUI5 library on SAP HANA database.

Indexing SAPUI5 Library

As discussed in the previous chapter, all the versions of the SAPUI5 library are available on the internal SAP Nexus Sonatype Repository. Therefore, indexing any version of the library is done with the help of the “**bower**” npm module.

Bower, being a package manager, is used for fetching and downloading the SAPUI5 library from the Nexus Sonatype repository to the project’s workspace. This is accomplished by first creating and configuring a “bower.json” file with the name and path of each version of the SAPUI5 library. Afterwards, by executing the following command: **bower install**, the versions of the library, which are specified within the “bower.json” configuration file, will be automatically downloaded and stored in the project’s workspace in a folder named “bower_components”.

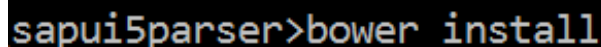
For example, the following figure 4.3 shows the configurations of the “bower.json” file.



```
1 {
2   "name": "sapui5Parser",
3   "version": "0.0.0",
4   "dependencies": {
5     "sapui5-1.44.8": "http://nexusrel.wdf.sap.corp:8081/nexus/service/local/repositories/build.milestones.hosted/content/com/sap/ui5/
6     "sapui5-sdk-1.38.9": "http://nexusrel.wdf.sap.corp:8081/nexus/service/local/repositories/build.milestones.hosted/content/com/sap/
7     "sapui5-1.28.9": "http://nexusrel.wdf.sap.corp:8081/nexus/service/local/repositories/build.milestones.hosted/content/com/sap/ui5/
8   },
9   "private": true
10 }
```

Figure 4.3: Example showing the configuration of the “bower.json” file

And by executing the following command shown in figure 4.4, the specified versions of the SAPUI5 library, within the “bower.json” file, will be downloaded to the “bower_components” folder.



```
sapui5parser>bower install
```

Figure 4.4: Executing bower install example

Maintaining SAPUI5 Library

In this part, a detailed explanation of how the SAPUI5 library is discussed. The first step required is to provide a proper model definition for the SAPUI5 library.

Model definition for the SAPUI5 library

The SAPUI5 library is very huge, that's why it is necessary to have a properly defined model with only the important information which will help in performing the upgrade check. As mentioned and explained in the previous chapter, this information includes the SAPUI5 version, UI Control library, Control name and its parent control class, function name, function type, description of this function as well as its return type and parameters. In addition, the accessibility of the function is also important for knowing if it is a private or a public one. Moreover, the coding of the control is essential in detecting whether there are changes made in the new version or not.

This is achieved by parsing the “/resources/sap” directory to fetch all the SAP UI control libraries ex: “sap/m”, “sap/ui/commons”,...etc.

HANA XS Backend

A package called “UpgradeCheck_SAPUI5”, as shown in figure 4.5, is created which groups together all the related information and definitions for the database schema.

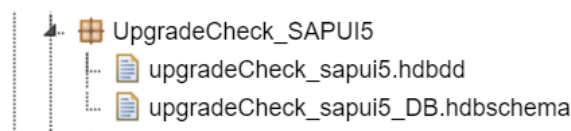


Figure 4.5: Package created on HANA XS

As shown in the figure 4.5, the following two files are created within the package:

- **upgradeCheck_sapui5_DB.hdbschema:** This file is used for defining the name of the schema “SAPUI5_Upgrade”
- **upgradeCheck_sapui5.hdbdd:** This file is used for defining the HANA entity table. The table “agsdev.UpgradeCheck_SAPUI5::upgradeCheck_sapui5.SAPUI5Lib” is created for maintaining the SAPUI5 library.

The following figure 4.6, is a diagram representation of the created table “SAPUI5Lib” on HANA database:

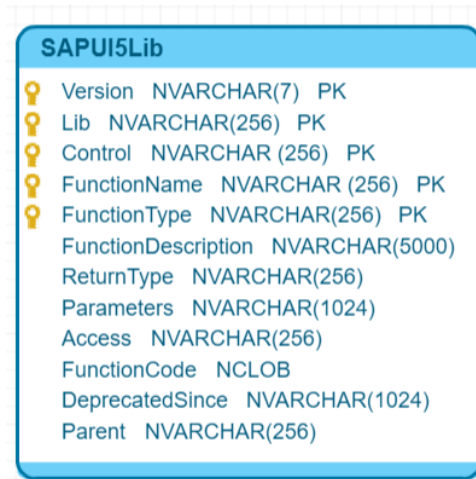


Figure 4.6: SAPUI5Lib table on HANA

LibStore.api.node.js

For maintaining the SAPUI5 library, the module “LibStore.api.node.js” gets executed. This module contains the path of the specific SAPUI5 library version to be stored. The connection to the SAP HANA database gets established by calling the “UpgradeHelper.db.node.js” module. Afterwards, the SAPUI5 library parser class “Lib.js” is called with the given path as a parameter, and thus the SAPUI5 library starts to get analyzed.

Lib.js

The “Lib.js” is used for parsing the given SAPUI5 library. It traverses the whole “/resources/sap” directory, considering only the debug files with the suffix “-dbg.js”.

Functions:

- **_traverseTree(pathToRootDirectory):** This function takes the path to the root directory as a parameter and it traverses the whole directory, checking only for the debug files with the suffix extension “-dbg.js”.

For each debug file, an instance of the “File.node.js” is created with the name, path, size, level and root name attributes. Also the function “_handleFile” is called for every debug file found.

- **_getRootName(pathToRootDirectory):** This function takes also the path to the root directory as a parameter and returns the root name of the directory.

Class Diagram

The below figure 4.7 shows a microscopic view of the “Lib.js” Class diagram (refer to figure 4.2 for the complete diagram):

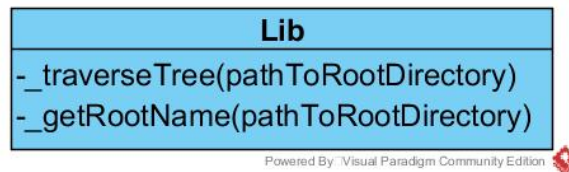


Figure 4.7: Microscopic view of Lib Class Diagram

File.node.js

As shown in figure 4.9 below (please refer to figure 4.2 for the complete diagram), this class holds all the information about any file. It has the following attributes:

- **_name:** This attribute is the name of the file
- **_path:** This attribute is the path of the file
- **_size:** the size of the specified file object
- **_level:** This is the depth of the node file in the directory
- **_pathStructure:** this attributes defines the structure path of the file
- **_fileData:** This attribute holds the file data content
- **_file_type:** This attribute defines the type of the file. Examples of the file type are JS, CSS, XML ...etc.
- **_sapui5_type:** This attribute is used during the parsing of the SAPUI5 source code. It defines the SAPUI5 type of the file, for example: Controller, Manifest, View .. etc

Functions

The following functions are implemented within the “File” class:

- **_getPathStructure(filePath, fileRoot):** This function takes the whole path to the file and the root directory and returns a new path structure with the root directory path removed from the path of the file.
- **_initFileType(fileName, filePath):** This function takes as parameters the file name, as well as its path. It returns the type of this file by checking the extension from its path. In addition to that, according to the file type the respective parser gets called. For instance, a “.js” file gets parsed by calling the “JS.node.js” and a “.xml” file gets parsed by calling the “XML.node.js”.
- **_initSapui5Type(fileName, fileData, filePath):** This function is used during parsing and analyzing the SAPUI5 test app. It takes as parameters the file name, file content and its path and checks for the SAPUI5 type of it. Furthermore, according to the SAPUI5 file type the respective parser gets called. At the end it returns the SAPUI5 type of this file.
- **getPathStructure():** This is a getter function, which returns the structure path of the file with the root path omitted.
- **getFileType():** This is a getter function returns the type of the file.
- **getSapui5Type():** This getter function returns the SAPUI5 type of the file.
- **getName():** This getter function returns the name of the file.
- **getPath():** This getter function returns the path of the file.
- **getSize():** This getter function returns the size of the file.
- **getLevel():** This getter function returns the depth of the file in the directory.

Furthermore, this class is also inherited by other modules according to the type each File as follows:

- **XML.node.js:** This module is responsible for parsing any XML file with extension “.xml”.
- **JS.node.js:** This module is responsible for parsing the JavaScript file type with extension “.js”.
- **Manifest.node.js:** This module is used during the SAPUI5 source code parsing phase. It is responsible for parsing the application descriptor file “manifest.json” within the given SAPUI5 application.
- **neo-app.node.js:** This module is responsible for parsing the application descriptor file “neo-app.json” within the given SAPUI5 application.

- **Component.node.js:** This module is also used during the SAPUI5 source code parsing phase. It is responsible for parsing the “Component.js” file within the given SAPUI5 application.

JS.node.js

This module is used for parsing any JavaScript file type. It is using the following two npm modules:

1. **parse-comments:** This module is used for parsing the comments within any JavaScript file [9].
2. **scopenodes:** This module is used for parsing any JavaScript file by fetching the outer program scope and its functions [20].

This module has some attributes which are defined as follows:

- **_rawcomments:** This holds the all the comments within the JavaScript file
- **_rawjs:** This attribute holds the output resulted from the “scopenodes” npm module.
- **_filePath:** This attribute holds the path of the JavaScript file
- **_methods:** This attribute is the output result of executing the function “_getMethods()”. It holds all the functions which are found within the JavaScript file
- **_importedControls:** This attribute is the output result of the function “_getImportedControls()” containing all the imported controls which are found within the JavaScript file.

Functions

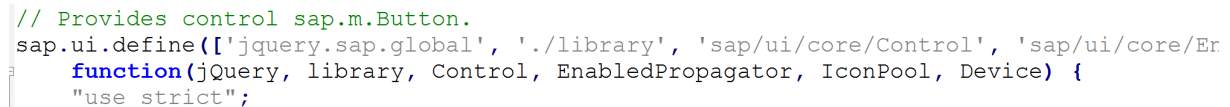
Within the “JS.node.js” module, the following functions are implemented:

- **_getMethods(rawJS, rawComments, filePath, fileData):** This function takes as parameters “rawJS”, “rawComments”, file path and file data attributes and returns the methods within the given JavaScript file. In addition to that, the model definition is created for each function within the UI control libraries in the SAPUI5 library.

- **insertData(methods, filePath, version):** This function is used for storing the SAPUI5 library.
- **_getImportedControls(rawJS):** This function takes the “rawJS” attribute as a parameter and returns the SAPUI5 controls which are imported within the given JavaScript file.

Specifically, the imported controls within any SAPUI5 file are found as parameters to the “sap.ui.define” function. As a result, this syntax is searched by the parser inside the JavaScript file.

For example the following figure 4.8 is a screenshot showing the imported controls within the “sap.m.Button” control:



```
// Provides control sap.m.Button.
sap.ui.define(['jquery.sap.global', './library', 'sap/ui/core/Control', 'sap/ui/core/Er
function(jQuery, library, Control, EnabledPropagator, IconPool, Device) {
    "use strict";
```

Figure 4.8: SAPUI5 imported controls within the JavaScript file

- **_getComment(method, rawComments, control, version):** This function takes the function name as a parameter and returns the comment associated with this function within the JavaScript file.
- **_getProperties(rawJS, rawComments, control, version):** This function takes as parameters the “rawJS” and “rawcomments” attributes, in addition to the control name and version number and returns the properties of the given control.
- **_addPropertiesComments(properties, controlProp, rawComments, control, version):** This function takes as parameters the list of properties, empty array, “rawcomments”, control name and the version number. It fills the empty array with the properties and their corresponding comments and returns it.
- **getMethods():** This is a getter function, which returns all the methods found within the JavaScript file.
- **getImportedControls():** This is a getter function, which returns all the imported controls within the JavaScript file.

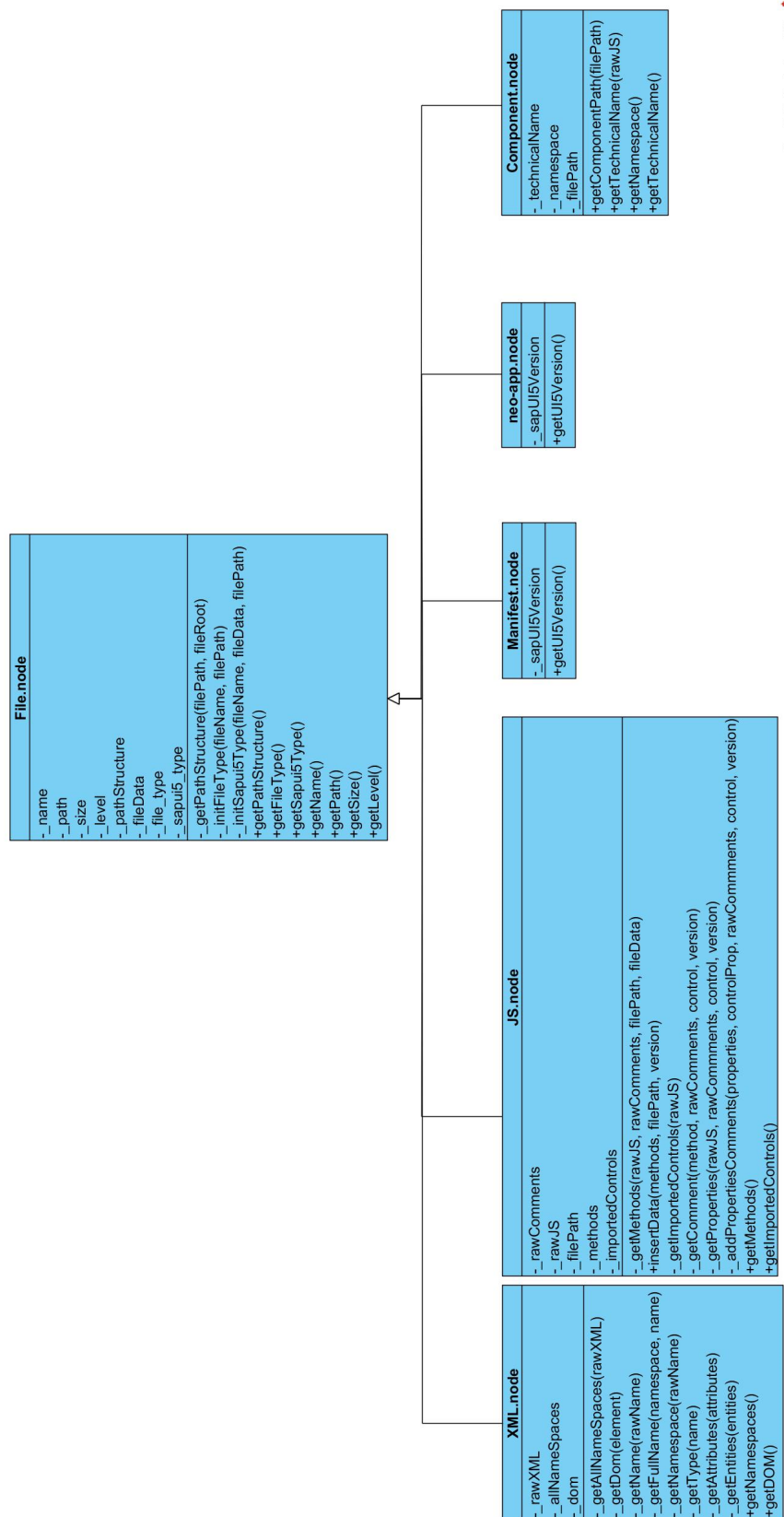


Figure 4.9: Microscopic view of the Class Diagram of “File.node”

ModelDefinition.node.js

This module is responsible for the creation of the model definition of the specific version of the SAPUI5 library. In other words, for each control library as well as their functions the model is defined by extracting the relevant information, which is needed for checking the upgrade risk impact. This is accomplished by implementing the “createModelDefinition” which gets called for each function within the JavaScript parser module.

UML class Diagram

The following figure 4.10 shows a microscopic view of the UML class diagram:



Figure 4.10: Microscopic view of the class diagram of “ModelDefinition.node”

4.2.3 Indexing SAPUI5 App Source Code

As discussed earlier, the source code of the existing SAPUI5 application is provided by the user via a URL to the Git repository. Consequently, cloning the Git repository is considered an essential first step for indexing the source code of the SAPUI5 application. This is achieved by configuring the Jenkins job, which will be explained later in section 4.3, to automatically clone the Git repository to its workspace.

4.2.4 Start.js

Obviously from its name and as shown in figure 3.4, this is the first node.js module to be executed for running the project. This module is called by Jenkins, which takes the following inputs as parameters:

- **appName:** This variable holds the name of the SAPUI5 application to be checked.
- **upgrade:** This is a boolean value, indicating whether to execute the Upgrade-Helper module or not. If it is set to true, the “init” function within the “Upgrade-Helper.api.node” module gets called.

- **docuGen:** This is also a boolean value, indicating whether to execute the Docu-Generator module or not. If it is true, then the “init” function within the ‘Docu-Generator.api.node’ module gets called.
- **testAppPath:** This variable holds the path to the cloned SAPUI5 source code to be investigated. This path is located on the server within the Jenkins workspace.
- **targetVersion:** This variable holds the target version of the SAPUI5 library to be checked for upgrading.
- **currentVersion:** This variable holds the value of the current SAPUI5 version of the given application.

4.2.5 Upgrade Helper

UpgradeHelper.api.node.js

This module which gets executed when the user checks on the respective checkbox “UpgradeCheck” via Jenkins interface. It holds the user input information needed for performing the upgrade check process.

init(appName, pathToApp, targetVersion, currentVersion): This function takes as parameters the name of the given application, the path to the source code of the SAPUI5 application, the target version to be checked for upgrade and the current version. The “builder.node.js” module gets called with the respective properties needed for executing the Upgrade check process.

Class Diagram

The following figure 4.11 is a microscopic view of its class diagram:

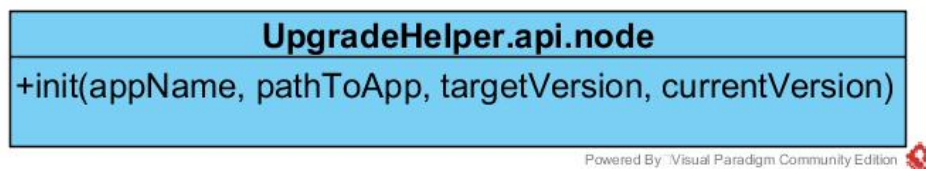


Figure 4.11: Microscopic view of the UML class diagram of UpgradeHelper.api.node module

builder.node.js

Indicated by its name, this module is used for a step by step building of the parser of the UpgradeHelper module in addition to its output report.

The following function is implemented within this module:

create(module, options, callback): This function takes as parameters the parser module, the properties of the application to be investigated. The init function of the respective parser module gets called (in this case “Upgrade.parser.node”). The “UpgradeHelper.parser.node.js” gets called with the cloned path of the application to be checked.

Class Diagram

The following figure 4.12 shows a microscopic view of the UML class diagram of the “builder.node” module:

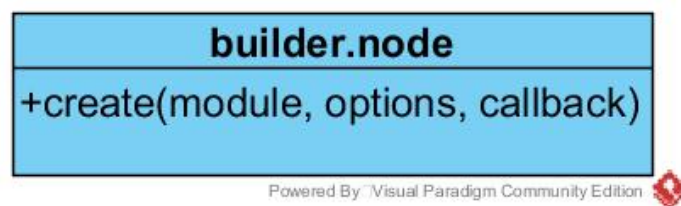


Figure 4.12: Microscopic view of the UML class diagram of builder.node module

UpgradeHelper.parser.node.js

Using this module, the path to the SAPUI5 application is passed as a parameter by calling the “Parser.js” class for parsing its source code.

Functions:

- **init(docu, callback):** This function takes as a parameter the general information of the SAPUI5 application to be investigated. In addition, it is used for fetching the current SAPUI5 version of the application, it was specified by the user as “Auto”.
- **generate(callback2):** This function is responsible for fetching the information, which should be available in the report document. For example, getting the list of deprecated controls, difference between both the current and target versions, ...etc.

- **get(callback):** This function is used to generate a word document with the output results. This is accomplished with the help of the npm module “docxtemplater”. This helper module is used for generating a “.docx” document from an already existing template, by changing any pattern “placeholder” inside the document with data.

Class Diagram

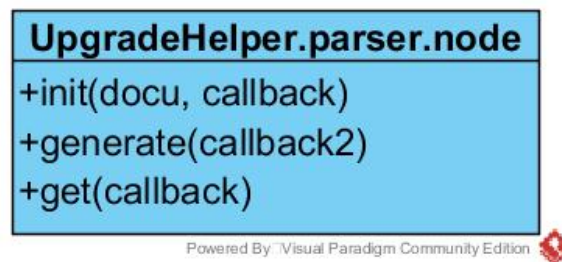


Figure 4.13: Microscopic view of the UML class diagram of UpgradeHelper.parser.node module

Parser

In this part, the SAPUI5 application gets parsed, so that all the required information needed for performing the upgrade check gets extracted.

Parser.js

Likewise the “Lib.js”, this class is considered the main module while executing the upgrade check process. This is because, this class is responsible for parsing the whole source code of the SAPUI5 application. In other words, the whole directory of the application’s source code is traversed by the “Parser.js”. For each file found in the directory, the function “**_handleFile(file)**” gets executed which creates an instance object of the class “File.node.js”. The function “**_initSapui5Type(fileName, fileData, filePath)**” gets called inside the “File.node.js” class, which takes as parameters the file name, path and content and then returns the SAPUI5 type of this file. A SAPUI5 file type can be a view, controller, component, manifest, ...etc.

As a result, some information returned by the parser can be helpful for the upgrade checking procedure. In addition, it can be a perfect indicator of the risk impact behind upgrading the SAPUI5 application to the target version.

Particularly, this information is listed as follows:

- **General information about the SAPUI5 app:** Using the parser, the following general information about the given SAPUI5 application can be retrieved:
 - **Namespace of the application:** The namespace of the given application is found within the application’s “Component.js” file. By calling the “getNamespace()” within the parser, the function “getNamespace()”, inside the “Component.node.js” parser module, gets called and the namespace is returned.
 - **Current version of the SAPUI5 app:** The SAPUI5 version of the given application is tremendously required, in order for having the capability to compare between both the current and target versions. Usually, the version of any SAPUI5 application is specified within the “manifest.json” file. The figure below 4.14 shows how SAPUI5 version is specified in the “manifest.json” file of the application.

```
"sap.ui5": {  
  "_version": "1.1.0",  
  "dependencies": {  
    "minUI5Version": "1.38.1",
```

Figure 4.14: manifest.json file containing the SAPUI5 version

As a result, getting the information about the SAPUI5 version of the given application is accomplished by parsing the “manifest.json” file and fetching the property “minUI5Version” under “sap.ui5.dependencies”. This is achieved by implementing the function “getUI5Version” within the “Parser.js” class. The parser checks if the application descriptor file “manifest.json” exists or not. If it exists, the function “getUI5Version()” gets called within the Manifest parser module “Manifest.node.js”, which returns the specified SAPUI5 version. Otherwise, the parser checks the “neo-app.json” file by calling the “getUI5Version()” function within the “neo-app.node.js” parser module, which extracts the specified SAPUI5 version.

- **OData Service used:** The information about the OData service used by the application is found within the application descriptor file “manifest.json” under “dataSources” property. This information includes the name of the OData service, its URL in addition to its version. The figure 4.15 below shows an example of OData service defined within the application descriptor file:

```

"dataSourcees": {
  "invoiceRemote": {
    "uri": "/destinations/northwind/V2/Northwind/Northwind.svc/",
    "type": "OData",
    "settings": {
      "odataVersion": "2.0"
    }
  }
}
}

```

Figure 4.15: OData service defined within the “manifest.json” file

Likewise parsing the SAPUI5 version, the Parser class first checks if the given application contains a “manifest.json” file within its root directory or not by implementing the function “getOData()”. If it exists, then the function “getODataService()” gets called within the “Manifest.node.js” parser module. Otherwise, the Parser checks the other application’s descriptor file “neo-app.json”, where the destination to the OData service is found under the “routes” property.

- **The views used in the application:** The function “_initSapui5Type()” first checks if the given file has an extension “.view.xml”, then it returns that it is a view SAPUI5 type. Subsequently, this view gets parsed by the respective XML parser module “XML.node.js”. At the end, the overall information about the used views is retrieved by implementing the function “getViews()” within the “Parser.js” class, which returns an array of all the application’s views.

XML.node.js

As mentioned before and from its name, it is responsible for parsing the XML views which have been used in the SAPUI5 application. This module is inherited from the class “File.node.js”. Information about the view name, its path, namespace, controls and their properties is extracted with the help of the npm module “xml-parser”.

This is accomplished by implementing the following functions:

- **_getAllNameSpaces(rawXML):** This function is used to get all the available nameSpaces in the specified view. It takes as a parameter the rawXML, which is the output resulted from the npm module “xml-parser”. It returns an object containing all the nameSpaces within the view, by fetching the attributes of the root of the XML of this view which start with “xmlns”.
- **_getDom(element):** This function takes as a parameter the root of the XML view and returns the view’s DOM object in terms of entities, their attributes and children. This is achieved by implementing and calling the following functions:

- * **_getNamespace(rawName):** This function takes as a parameter the name of the parsed root of the XML. It returns the name of the namespace, by searching against the namespaces object returned from “_getAllNameSpaces(rawXML)” function. For instance “core:” would return “sap.ui.core” as a namespace.
 - * **_getName(rawName):** This function takes as a parameter the name of the node found while parsing the XML tree. In other words if the name is preceded by the namespace, the function returns only the name. For example calling the function on “mvc:View” would return “View” as the name.
 - * **_getType(name):** The function takes as a parameter the name returned from “_getName(rawName)” function and returns the type of it. Basically it checks if the name starts with an uppercase, then it is of type control otherwise it is of type aggregation.
 - * **_getFullName(namespace, name):** This function takes as parameters the namespace and the name of the control and returns the full name of the control. For example, “sap.m” as a namespace and “Button” as a control name would result in “sap.m.Button” as the full name returned.
 - * **_getAttributes(attributes):** This function takes as a parameter the attributes of the given node within the XML tree. An object is created for each attribute by classifying it into “key” and “value”. Finally, the function returns a list of objects with all the attributes of the given node.
 - * **_getEntities(entities):** This function takes as a parameter the entities found for a given node, while parsing the XML tree of the view. Likewise the “_getAttributes(attributes)” function, an object is created for each entity with “key” and “value” as properties. Afterwards, the function returns a list of entities objects of the given node.
- **getNamespaces():** This is a getter function, which returns the object containing all the namespaces within the view.
 - **getDOM():** This is a getter function, which returns the DOM of the XML view.
- **The controllers used in the application** The information about the names of the controllers used within the SAPUI5 application and their content is extracted. This is accomplished by calling also the “_initSapui5Type()” inside the “_handleFile()” function, which checks for the files with the extension “.controller.js” and then controller is returned as a SAPUI5 file type. In addition, it gets parsed by the respective JavaScript parser “JS.node.js”.
- Getting an overall information about all the used controllers within the application is done by implementing the function “getControllers()” within the “Parser.js” class, which returns an array of all the application’s controllers.

Some information like the control name, path, functions and their parameters and return type as well as imported controls is extracted from the “JS.node.js” module.

- **Standard controls used within the application:** Getting an overview information about the SAPUI5 standard controls, that have been used in the application, is crucial while performing the upgrade check procedure. Using the “Parser.js” class, this information is fetched from two different SAPUI5 file types:

1. **From the controllers used within the application:** Controls which are imported are defined at the beginning of each controller. As mentioned earlier in this chapter, the function “_getImportedControls(rawJS)” within “JS.node.js” returns a list of all the imported controls inside each controller.

In addition to that, in order to get all the information about the standard controls from the whole application perspective and not only in one controller, the following functions are implemented within the “Parser.js” class:

- **getJS():** This function returns a list of all the JavaScript files in the application.
- **getJSInfo(JSfiles, type):** This function takes as parameters the list of JavaScript files, in addition to the SAPUI5 file type ex: “js”, “controller”, “object”,...etc. It iterates over all the JavaScript files of type controller and for each controller, the following functions get called:
 - * **_getControls(JSfile, type):** This function takes the JavaScript file and its SAPUI5 type as parameters and returns the controls found in this file by calling the “getImportedControls()” in “JS.node.js” module. In addition to that, the function keeps track of all the controls used within the whole application by adding each control found in the JavaScript controller to the global array “this._appControls”.
 - * **_getMethods(JSfile, type):** This function also takes as parameters the JavaScript file as well as its SAPUI5 type and returns the list of methods found within each file. This is achieved by calling the “getMethods()” inside “JS.node.js” module.

2. **From the views used within the application:** The information about the standard controls which are used in the application, can also be extracted by parsing the XML control tree in the view. This is achieved by the implementing the following functions within the “Parser.js”:

- **getViews():** This function returns a list of all the views found within the SAPUI5 application.
- **_getViewInfo(views):** This function takes as a parameter the list of views found within the applications and returns a list containing all the information about each view. This is accomplished by implementing the following functions:

- * **_getXMLNameSpaces(nameSpaces):** This function takes as a parameter the nameSpaces returned from calling the function “getNamespaces()” inside “XML.node.js”. It returns a list of objects with the information about each nameSpace found in the view.
- * **getXMLTree(viewName, dom, pathStructure, viewPath):** This function takes the view name, the path and the DOM and returns the XML control tree of the view. In other words, it fetches the controls as well as their properties by traversing every node in addition to its children in the XML view. In addition, the function also keeps track of all the controls used within the whole application by adding each control found in the XML view to the global array “this._appControls”.

Finally, by calling the function **getAppControls()** the list of all the controls (this._appControls), which have been used in the application is returned.

- **Properties used within each control:** These properties can be fetched from the XML views used within the application. Besides getting information about the used controls, the function “getXMLTree(viewName, dom, pathStructure, viewPath)” provides also information about the properties of the used controls in the view. This is achieved by fetching the “attributes” property for each control found within the XML view. In addition, the global array “_appControlProperties” is filled to keep track of all the information about all the control properties used within the whole application.
- **Analysis of the occurrence of each control within the application:** Besides helping the users in knowing the most used control within the application, this information plays an important role in determining the impact analysis on performing an upgrade to the existing SAPUI5 application. It also plays a big role in calculating the risk factor behind performing an upgrade.

The following function is implemented: **getControlsAnalysis(appControls):** This function takes as a parameter the list of SAPUI5 standard controls used in the whole application. By looping over each control, the occurrence of each control is calculated by determining the locations where each it has been used. This is accomplished by using the following helper functions:

- **checkIfExists(result,control):** This function takes as parameters the array of used controls in addition to the control name. It returns a Boolean determining whether this control already exists in the array or not.
- **getLocations(control):** This function takes the control object as a parameter and returns a list of object containing the occurrences of this control by calling getOccurenceInFile() function.

- **getOccurrenceInFile(controlName,fileName):** This function takes as parameters the name of the control in addition to the name of file it was used. It returns at the end the number of times this control has been used in the file.
- **removeDuplicateFiles(locations):** This function takes as a parameter the array of the calculated locations of a given control. It checks that there are no duplicates within this array and remove them if they exist.

Consequently, this information can be used in performing the following tasks:

1. **Deprecated standard controls and properties:** Identifying a list of deprecated standard controls is considered a powerful indicator for the users willing to upgrade their applications. This is because having such a list is very helpful on knowing whether the upgrade will be safe or it can result in some compatibility issues with the new version, which can lead to application failure. The information about the deprecated standard controls as well as the properties is retrieved by implementing the following functions:
 - (a) **checkDeprecatedControls(controls, targetVersion, callback):** This function returns a list of the deprecated standard controls and takes as parameters the list of used standard controls within the SAPUI5 application in addition to its target SAPUI5 version. The connection to the database is established and by iterating over all the controls, the function “checkDeprecated()” within the “UpgradeHelper.db.node.js” module gets called for each control in the list.
 - (b) **checkDeprecatedProperties(appProperties, deprecatedControls, targetVersion, callback):** This function takes as parameters the list of properties used within the application as well as the list of standard controls and the target version for the application to be upgraded to. It returns the list of deprecated properties, by iterating over the list of properties and for each property the function “checkDeprecatedProperties()” within the “UpgradeHelper.db.node.js” module gets called.
2. **Comparison between current and target versions:** By performing this step, users are able to get information about the changes made in the new version of the SAPUI5 library. Those changes are classified into two types as follows:
 - **Added changes:** Those are the changes that has been newly added to the target SAPUI5 version, in which the user wants to upgrade to.
 - **Removed changes:** Those are the changes that exists in the current SAPUI5 version of the application but has been removed from the target SAPUI5 version to be upgraded to.

This is accomplished by implementing the following function:

- **checkDifferences(version, toVersion, callback):** This function takes as parameters the current and target SAPUI5 versions and returns an array of the changes applied to each control used in the application. By iterating over all the controls and for each control, the source codes of both the current and the target SAPUI5 versions are fetched by calling the “findDiff()” function inside the “UpgradeHelper.db.node.js”.

In addition to that, the function compares both source codes and checks if there any differences between them. A difference between the source codes would imply that there are changes applied to the SAPUI5 library in the new version release compared to the current one. The npm module “diff” is used for comparing both source codes and detecting the changes made.

- **separateLib(control):** This function is used for splitting the library and control name from the full control name. For instance, “sap.m.Button” would result into “sap.m” as library name and “Button” as the control name. In other words, it takes as a parameter the full name of the control and returns a string of the library name.

3. **Calculation of the amount of changes made:** The next step after fetching the changes applied to each standard control in the new version of the SAPUI5 library, the percentage of the amount of these changes is then calculated. As a result, the following formula 4.1 is used in the calculation:

$$\text{Changes \%} = \frac{\text{total number of changed lines}}{\text{Total number of the old version lines}} \times 100 \quad (4.1)$$

4. Risk Factor Calculation:

In addition to calculating the percentage of changes made to each control in the new version, a risk factor is also calculated. The risk factor is a proposed Key Performance Indicator (KPI) for showing the users an indication of the impact analysis, which would result from this control when they upgrade their application.

“calculateRiskFactor(control, lib, percentage, controlsAnalysis)”: This function is called inside the “checkDifferences()” function. It takes as parameters the full name of the standard control, library name, in addition to the calculated percentage of changes and the list of occurrences of the used controls in the application. As explained by the below equation 4.2, this function calculates and returns the risk factor of the given control.

$$\text{Risk Factor} = \frac{\text{Changes \%}}{10} \times \text{Control occurrence} \quad (4.2)$$

where 10 is a constant for getting a reasonable range of risk factor value

5. **Upgrade Impact Analysis:** Getting an overview about the impact analysis on performing an upgrade to the given SAPUI5 application is retrieved by the function:

getImpactAnalysis(): This function returns an array of objects, containing the overall impact analysis information about each control used within the application. This information includes: Control name, percentage of changes made, number of occurrence within the application and the calculated risk factor.

Class Diagram of “Parser.js”

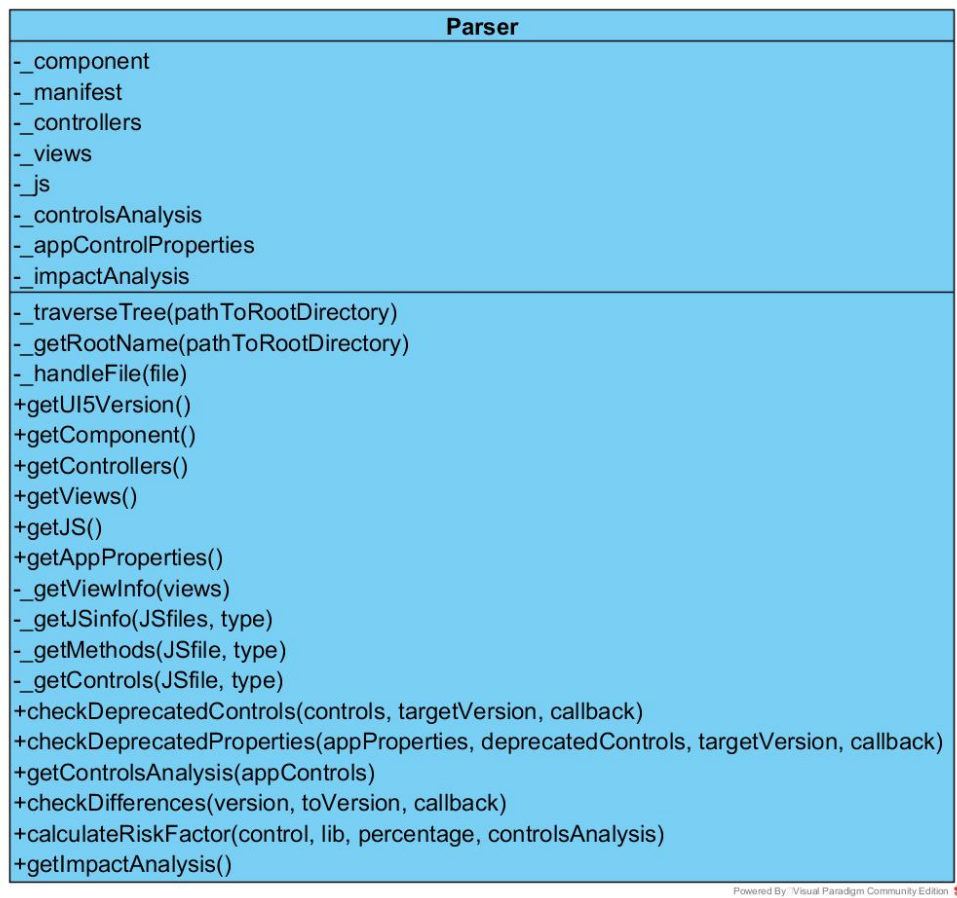


Figure 4.16: Microscopic view of “Parser.js” class diagram

UpgradeHelper.db.node.js

This module is implemented for dealing with the database in terms of establishing database connection and executing SQL statements. The npm module “hdb” is used for allowing the communication with SAP HANA database.

Functions:

- **on(client):** This function takes as a parameter the instance of the SAP HANA client. It is used to check the connection to SAP HANA database.
- **connect(client):** This function also takes as a parameter the instance to the SAP HANA database client. It is used for establishing the connection with the database.
- **select(client, command):** This function takes as parameters the instance of the database client and the SQL query. It is used for executing SQL select statements with the given query.
- **insert(client, data):** This function is used for storing the SAPUI5 library. It takes as parameters the client instance and the data to be inserted.
- **checkDeprecated(toVersion, lib, control, fileName, filePath, pathStructure, callback):** This function is called inside the “checkDeprecatedControls()” function within the “Parser.js” class. It takes as parameters the target SAPUI5 version, the library name, control name, file path and the path structure inside the SAPUI5 application directory.

The function first checks if the control is an existing SAPUI5 standard control by ensuring that it exists in the database table. This is accomplished by executing a SQL SELECT query to the database table with the given control information. Afterwards, another SQL SELECT statement is executed which checks the column “Deprecated” of the given control of the target version. In other words, if the “Deprecated” value is not empty then this control is said to be a deprecated control otherwise it is not. In addition to that, the function “getMatches()” gets called which helps in determining the lines in the code, in which this deprecated control is used. Finally an object of the deprecated control is returned by the function.

- **checkDeprecatedProperties(toVersion, lib, control, property, fileName, filePath, pathStructure, deprecatedControls, callback):** This function takes as parameters the target version, library name, control name, property name, file name, file Path, path structure inside the SAPUI5 application directory and the list of deprecated controls.

The function checks first if the control is found among the list of deprecated controls. If it is found, then there is no need to check its property since its control is

deprecated. Otherwise, the function proceeds with checking if this property is deprecated or not. Like what was done in the “checkDeprecated()” function, it checks also if this property exists in the SAPUI5 library table for the given control or not. This is achieved by executing a SQL SELECT statement to the database with the given property information.

After that, if this property is existed the function proceeds with another SQL SELECT query to extract the ”Deprecated” column of this property. If the ”Deprecated” value is non empty then this property is deprecated. Moreover, the “getMatches()” function is also called to fetch the lines in the code, in which this property is used. Finally, the function returns an object of the deprecated property of the given control.

- **getMatches(control, fileName, dirPath, callback):** This function takes as parameters the pattern to search for the control or the property, file name and the path in the SAPUI5 application directory. It searches for the given pattern of the control/property inside the whole file and returns an array of objects with the line number and code where it was found. This is done with the help of the npm module “rx-text-search”.
- **findDiff(control, version, toVersion, lib, callback):** This function is called within the “checkDifferences()” inside the “Parser.js” class. It takes as parameters the control name, current SAPUI5 version, target SAPUI5 version and the library name. It returns an array of both the source code of the current and the target version of the given control. This is achieved by executing a SQL query which selects the “FunctionCode” attribute of both versions.
- **end(client):** This function takes as a parameter the instance of the SAP HANA client. It is used for closing the database connection.

Class Diagram

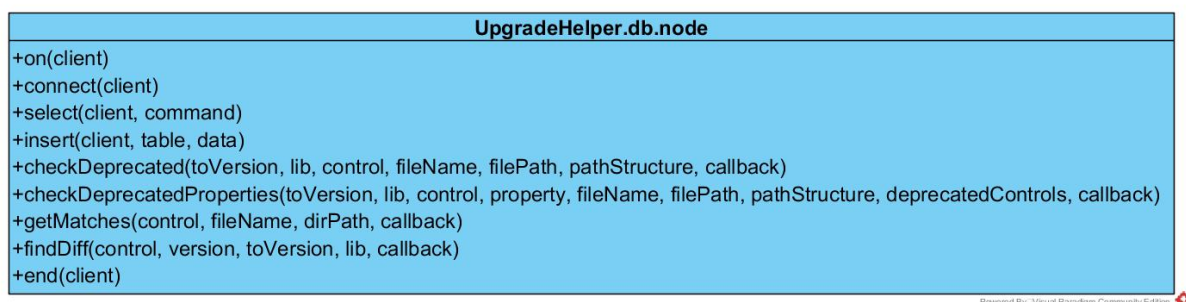


Figure 4.17: Microscopic view of “UpgradeHelper.db.node” class diagram

4.2.6 Documentation Generator

As mentioned earlier, the core SAPUI5 parser is a powerful base, which can be reusable for supporting additional modules. In this section, the core SAPUI5 parser is reused in generating a technical documentation about any SAPUI5 application. This is accomplished by having a another module called DocuGenerator.

DocuGenerator.api.node

As shown in the flowchart in figure 3.4, The DocuGenerator module gets executed when the user checks on the respective checkbox “DocuGen” via Jenkins interface.

init(appName, pathToApp, currentVersion, callback): This function takes as parameters the inputs needed for the Documentation Generator, which are the name of the application, the path to its source code on Jenkins and the current SAPUI5 version. It calls the “builder.node” module by passing the respective Documentation Generator parser module and the inputs as parameters.

Class Diagram

The following figure 4.18 shows a microscopic view of the UML class diagram of this module:



Figure 4.18: Microscopic view of “DocuGenerator.api.node” class diagram

DocuGenerator.parser.node

This module is responsible for creating the output report documentation. In addition, it is used for calling the core SAPUI5 parser with the given path to the SAPUI5 application’s source code.

Functions

Like the “Upgrade.parser.node” module, the same functions are implemented. However, the **generate(callback2)** function fetches more information, by calling other functions within the “Parser.js” class. For instance, a technical documentation document would

include more information about the application regardless of the ones used for checking the upgrade impact. In other words, it fetches for example the information about the functions used within the controllers, control tree of the xml views, ...etc.

Class Diagram

The following figure 4.19, shows a microscopic view of the UML class diagram of this module:

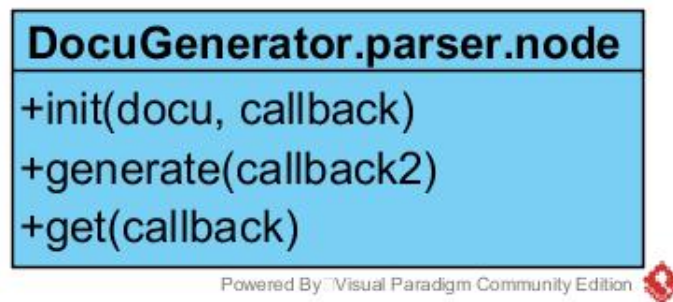


Figure 4.19: Microscopic view of “DocuGenerator.parser.node” class diagram

4.3 Jenkins Job Configuration

As mentioned earlier, Jenkins is used for ensuring CI. Besides, it is used in this thesis as a client side, by allowing the user to use its user interface in providing the inputs required for analyzing and checking the upgrade impact on their given applications.

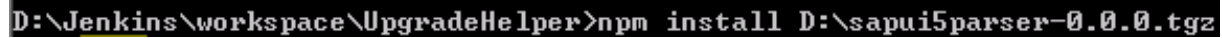
In this section, a step by step explanation on how Jenkins was configured and integrated with the Node.js backend project is discussed.

4.3.1 Packaging & installing the Node.js project on the server

The Node.js project needs to be deployed on the server where Jenkins is located. This is achieved by performing the following two steps:

1. **Packaging:** By executing ‘npm pack’ command on the root directory of the project, the Node.js project is converted into a so called npm package. This package is a compressed folder with the format: **<name>-<version>.tgz**

2. **Deployment and installation:** The npm package is then deployed on the server, where Jenkins is running on. Finally, the package can be installed on the created Jenkins job using the command shown in the following figure 4.20



```
D:\Jenkins\workspace\UpgradeHelper>npm install D:\sapui5parser-0.0.0.tgz
```

Figure 4.20: Installing the npm package on the Jenkins job

4.3.2 Configuring a Jenkins job

As previously explained in section 2.2.4, a Jenkins job project, called “UpgradeHelper”, is created by specifying the following configurations:

1. **This project is parameterized:** By configuring this, Jenkins provides a simple interactive user interface allowing users to provide the needed inputs. Specifying those inputs is done by configuring the following parameters within the Jenkins job:
 - **App name:** This is a string parameter for storing the value of name of the given SAPUI5 application.
 - **UpgradeCheck:** This is a boolean parameter that is represented to the users by a checkbox, where they can select whether they want to check the upgrade compatibility of their application or not.
 - **DocuGen:** Like the UpgradeCheck parameter, this is also a boolean parameter, where users can select whether they want to have a generated technical documentation for their application or not.
 - **Git_Repository:** This parameter is a of type string, which is used to store the URL to the Git repository, where this application is located.
 - **Target version:** This parameter is called “Choice parameter”, which is represented to the users by a dropdown list, where they can select the target SAPUI5 version to be checked for upgrade.
 - **Current SAPUI5 version:** This is also a “Choice parameter”, which allows users to select the current SAPUI5 version of their application. Among the dropdown list items there is an “Auto” selection, which means that the SAPUI5 version will be automatically determined by parsing the “manifest.json” application descriptor file.

The following figure 4.21 shows how the Jenkins user interface looks like, when clicking on “Build with Parameters”, after configuring the job as parametrized:

The screenshot shows the Jenkins web interface for configuring a build job named 'Project UpgradeHelper'. The left sidebar contains navigation links like 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build with Parameters', 'Delete Project', 'Configure', 'Git Polling Log', 'Move', and 'Test Results Analyzer'. The main area displays the configuration for the job, which requires several parameters: 'App name' (a text input), 'UpgradeCheck' (a checkbox that is checked), 'DocuGen' (a checkbox that is unchecked), 'Git_Repository' (a text input), 'Target version' (a dropdown menu showing '1.36.14'), and 'Current SAPUI5 version' (a dropdown menu showing 'Auto'). A 'Build' button is located at the bottom right of the configuration area.

Figure 4.21: The user interface of Jenkins Build with Parameters

2. **Git as a SCM:** The next step is the “Source Code Management” configuration section, which is responsible for fetching the source code of the application from the Git repository. This is accomplished by selecting “Git” and providing the pre-entered Git repository URL input. In other words, Jenkins fetches the source code and clones it to its job’s workspace. The following figure 4.22 shows a screenshot of configuring the source code management on Jenkins:

The screenshot shows the 'Source Code Management' configuration section in Jenkins. It has two radio buttons: 'None' and 'Git', with 'Git' selected. Below the radio buttons, there are two main sections: 'Repositories' and 'Branches to build'. The 'Repositories' section contains a 'Repository URL' field with the value '\$Git_Repository', a 'Credentials' dropdown menu with the value '- none -', and an 'Add' button. There are also 'Advanced...' and 'Add Repository' buttons. The 'Branches to build' section contains a 'Branch Specifier (blank for \'any\')' field with the value '*/master' and an 'Add Branch' button. At the bottom, there is a 'Repository browser' dropdown menu with the value '(Auto)'.

Figure 4.22: Configuring the Git repository URL under Source Code Management

3. **Poll SCM:** Afterwards, how the build job should be triggered is configured in section “Build triggers”. In this solution, it is necessary to keep track of the given application and this build job should be triggered whenever a change in the code

has been made (new commit). To achieve that, the “Poll SCM”, as shown in figure 4.23, is checked and scheduled to check for the code changes every minute by typing the “* * * * *”, which means that the schedule is every minute.

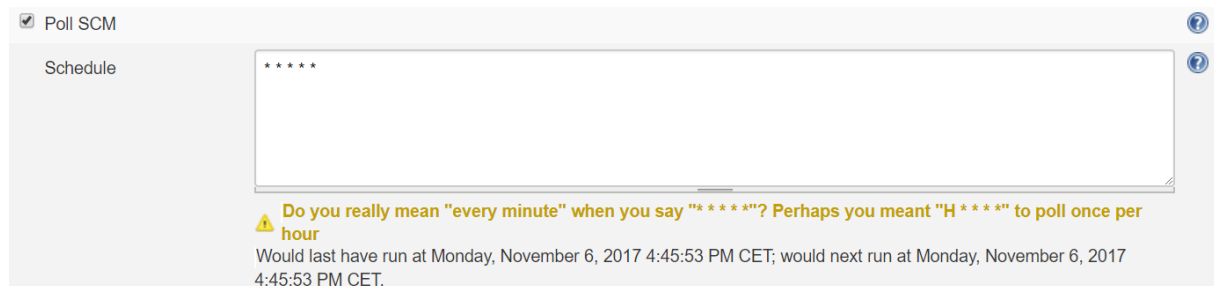


Figure 4.23: Poll SCM configured under Build triggers section

4. **Build:** This is the most important step in the configuration, since all the backend Node.js scripts get executed as a “Build step”. The “Start.js” Node.js module gets called by Jenkins with the respective inputs as a “Windows batch Command” as shown in the following figure 4.24:

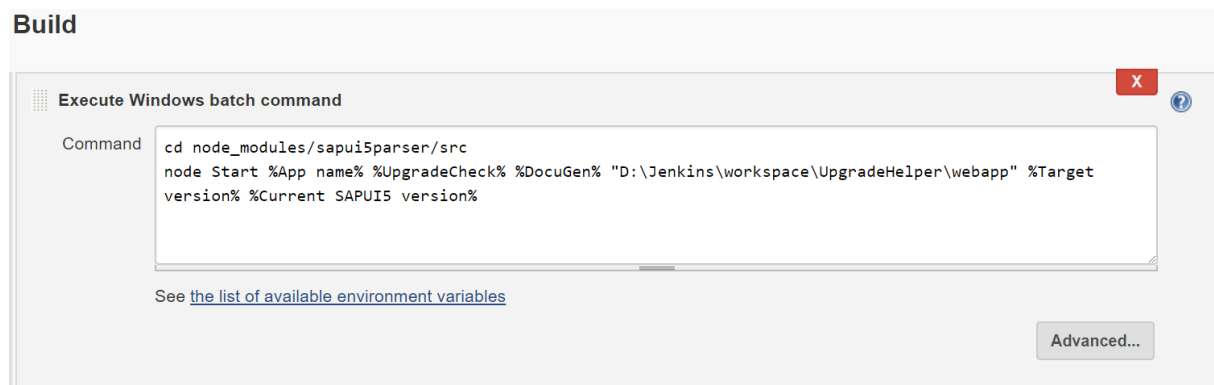


Figure 4.24: Jenkins Build Step

5. **Post-build Actions:** Finally, the output result documents are archived on Jenkins. This is achieved by selecting ‘Archive the artifacts’ as a “Post-build Action” as shown in the below figure 4.25:

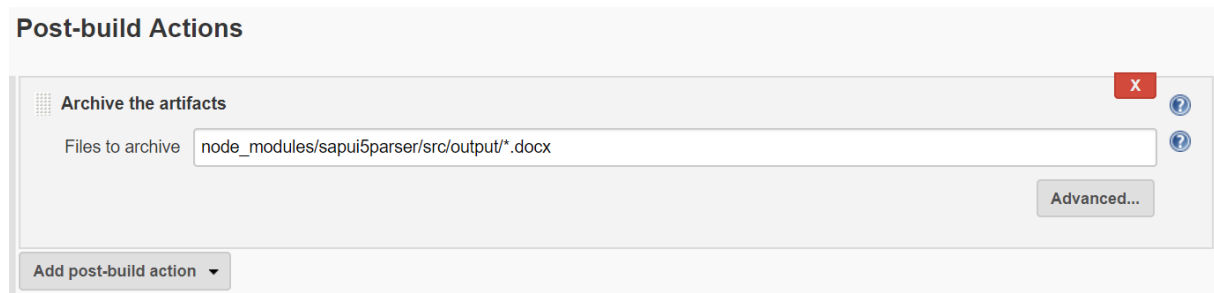


Figure 4.25: Archiving artifacts as a Post-build action

Finally, the archived artifacts out documents can be found on the Jenkins job's page under “Last Successful Artifacts”, as shown in figure 4.26 below:

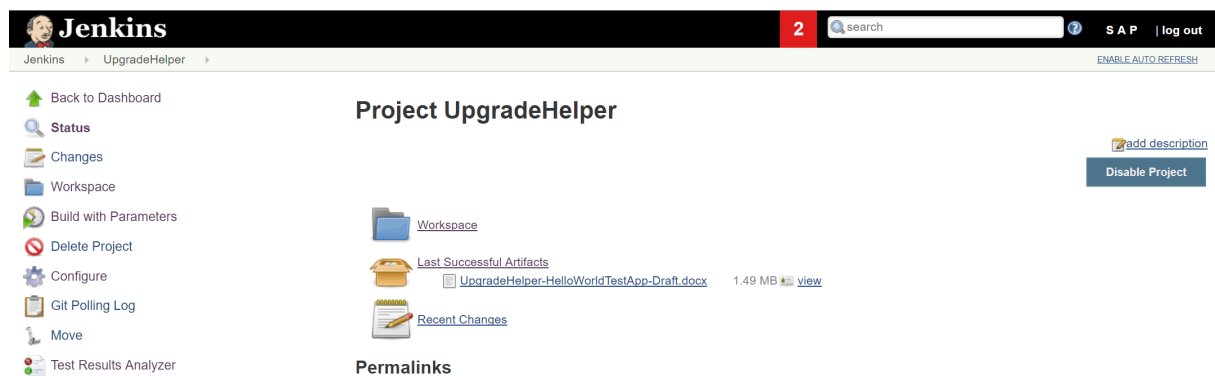


Figure 4.26: Jenkins archived artifacts

Chapter 5

Evaluation & Testing

In this chapter, a brief description about the output results is discussed. This is followed by an overview explanation of how the solution was tested.

5.1 Discussion of the output results

As mentioned earlier, the output is a generated word document “.docx” report according to the module executed. For instance, selecting both “UpgradeHelper” and “DocuGen” results in the generation of two separate word documents.

Using an existing template document for each module separately, the output results are filled within this template with the help of the npm module “docxtemplater”. According to each module (whether UpgradeHelper or DocuGenerator), the following two documents are generated:

1. **UpgradeHelper-<appName>-Draft.docx:** This document is generated on executing the UpgradeHelper module. This report consists of the following sections (example of report output is in appendix):
 - **General:** This section contains an overall general information about the application such as the SAPUI5 version, the namespace and OData service used.
 - **Upgrade Impact Analysis:** In this section, the risk analysis of the used controls is maintained in a list. This list is sorted descendingly from high risk controls to lower risk safe controls. For each control, the following information is listed in columns:
 - **Control:** This column holds the full name of the SAPUI5 control, for example “sap.m.Button”.

- **Modified by:** This column holds amount in percentage, in which this control has been changed.
- **Occurrence:** This column holds the information about the number of times this control has been used within the whole application.
- **Risk Factor:** This column holds the calculated risk factor of this control on upgrading the SAPUI5 application to the specified target version.

The following figure 5.1 shows an example of the upgrade Impact analysis on upgrading an existing SAPUI5 application from version 1.30.0 to 1.44.8. As shown in this figure, the top riskiest control is the “sap.ui.core.mvc.View” which has a risk factor value of 2509.78 since it has been modified by 50.20% and has been used 5 times in the application. On the other hand, the “sap.m.Page” control is less risky with a risk factor value of 1285.71, since it has been changed by 64.29% but only used two times throughout the application.


Control / Class	Modified by	Occurrence	 Risk Factor
sap.ui.core.mvc.View	50.20%	5 time(s)	2509.78
sap.ui.core.mvc.XMLView	118.53%	2 time(s)	2370.61
sap.m.Button	44.78%	5 time(s)	2239.22
sap.ui.core.mvc.Controller	52.31%	4 time(s)	2092.31
sap.ui.model.json.JSONModel	44.10%	3 time(s)	1322.92
sap.m.Page	64.29%	2 time(s)	1285.71

Figure 5.1: Upgrade Impact Analysis example

- **Deprecated Standard Controls:** This section lists all the information about the used controls that have been deprecated in the new version.
 - **Deprecated Properties of Standard Controls:** This section lists all the information about the properties of standard controls that has been deprecated in the new version.
 - **Analysis of used Standard Controls:** This section lists all the occurrence of each control used in the application, in addition to the files where they are used.
2. **TechnicalDocumentation-<appName>-Draft.docx:** This document is generated on executing the DocuGenerator module. The document contains general

information about the application, like the UpgradeHelper report, in addition to the following information:

- **Component:** This section contains all the information regarding the “Component.js” file.
- **Views:** This section contains information about each view used within the application. The information includes the view name, namespaces defined and a control tree of the used controls within the view.
- **Fragments:** This section contains information about the each fragment used within the application. Like in the views section, the same information is presented for each fragment used.
- **Controllers:** This section contains the information about each controller used within the application like its name, function used and imported controls.
- **JS:** This section is similar to the controllers section. However, it is not only limited to used controllers but to all the JavaScript files used in the application.

5.2 Testing

Testing is considered one of the important phases during the development life cycle. The solution was tested for ensuring the validity and stability of the solution and that all the requirements are fulfilled. As a result, testing was done in different ways as follows:

- **Functional Testing:** This involved the use of unit tests for testing each functionality of the solution individually. In addition, according to each use case, integration tests are performed for each module.
- **SAPUI5 Test Applications:** Multiple SAPUI5 applications with different SAPUI5 versions were used as test example inputs. The aim was to test the solution and specifically the UpgradeHelper module. The testing involved all possible combinations of inputs on various number of existing SAPUI5 applications. This is to verify the stability of the solution without any bugs.

Chapter 6

Conclusion

In this chapter, the work of the thesis is summarized. Finally, the limitations of the current work are highlighted and recommended future work is described.

6.1 Summary

In conclusion, the use of web applications is rapidly growing worldwide over the past years. As a result, users tend to move towards the use of modern web applications instead of the native ones.

The aim of this thesis was to develop an automated integration system for SAPUI5 web applications. The solution is intended to assist users, by helping them to know in advance the risk impact which can result from upgrading their SAPUI5 applications. This was achieved by first identifying the challenges that arise from using the JavaScript language, because of its dynamic and weakly typed nature. In addition, the thesis also states the problem of software maintenance in general and SAPUI5 in particular being prone to errors and compatibility issues, when it comes to upgrading from one version to the other.

According to the research made, the solution was designed by defining the list of the needed requirements, in addition to the different use cases that should be taken into consideration during the implementation.

Afterwards, the SAPUI5 library is indexed from the internal SAP Nexus Sonatype repository and through defining a proper model definition, the SAPUI5 library is maintained by storing it onto the SAP HANA database. The solution involved also Jenkins, with its build with parameters user interface for providing inputs. By properly configuring the Jenkins job, the respective Node.js modules get executed. By parsing and analyzing the source code of the given SAPUI5 application, the following information gets extracted:

- The current version of the SAPUI5 app if it is not already specified by the user in advance.
- The views which have been used in the application, by parsing their XML coding
- The controllers which have been used in the app, by parsing their JavaScript coding
- The functions which have been used within the JavaScript controllers
- The standard controls and the properties used within the application
- Analysis of the occurrence of each control within the app
- The amount of changes made in the new target version of the SAPUI5 library, in terms of number of changed lines
- The List of deprecated standard controls as well as the deprecated properties

Accordingly, the upgrade risk can be determined by calculating the risk factor using the previously defined KPI (refer to equation 4.2). Finally, a report is generated, which acts as an indicator for the users on the risks of upgrading their application. It guides them whether it is safe to upgrade to the new target version or stay with the current one until the compatibility issues are resolved.

Moreover, the implemented SAPUI5 core parser can be reusable for supporting additional modules. For example, it can be used for generating a technical documentation report for the given SAPUI5 applications.

6.2 Optimization potentials

There exist some limitations which are encountered by this solution. Due to the large size of the SAPUI5 library, the calculated performance time for parsing and inserting the whole library into HANA database takes approximately 15 minutes. As a result, the indexing and maintaining of the library is done manually using the “libStoring” module and not via Jenkins. Moreover, the solution only supports only one way for indexing of the source code of an existing SAPUI5 application, which is by providing the URL to the Git repository as an input. These limitations can be optimized in the future enhancement of the solution which will be explained in the next section.

6.3 Future Work

In this section, some proposed ideas are listed and explained, which can be considered as a future enhancement of this thesis.

6.3.1 Extending SAPUI5 Source Code Indexing Methods

Currently as previously mentioned, the solution supports one way of indexing the source code of the SAPUI5 application, which is using the Git repository URL.

Additional SAPUI5 source code indexing ways can be also supported in the future, such as the following:

- **Zip upload:** The source code can be provided by asking the user to upload it as a `.zip` file.
- **Nexus Sonatype Repository:** Some existing SAPUI5 applications are stored within the Nexus Sonatype repository. Consequently, the application can be also indexed by providing the URL to the repository where its source code is located.
- **ABAP BSP:** Some SAPUI5 applications can be also found on the ABAP BSP, therefore it is highly recommended to support the indexing of their source code through the integration with the ABAP BSP system.

6.3.2 Further Modules & Use Cases

As mentioned earlier, the core SAPUI5 parser can be reusable to support additional modules and use cases. For example, the solution can be used to help users ensure they are following the SAPUI5 best-practice guidelines. Consequently, an additional module “Code Review” can be implemented for generating a code review report for the given SAPUI5 application.

6.3.3 Parsing additional file types

This solution can be also improved by extending the parser to support other additional file types. For instance, HTML parser would be required for parsing the “index.html” file of the SAPUI5 application. Also, a CSS parser would be needed for parsing any CSS file associated with the application.

6.3.4 Remote triggering of Jenkins via a SAPUI5 application

The solution can be enhanced to be more interactive by building a SAPUI5 web application as a client side to be used by SAP customers. The idea of having a simple user friendly interface plays a big role in ensuring the validation of user inputs. Furthermore, the application should trigger Jenkins remotely with the required inputs as parameters. This can be achieved by configuring the Jenkins job to be triggered remotely, as shown in the below figure 6.1:

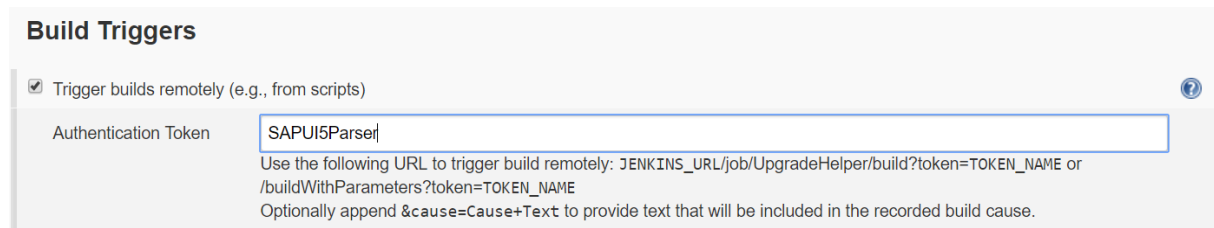


Figure 6.1: Configuring the remote triggering of Jenkins

Afterwards, the inputs can be passed from the SAPUI5 app by triggering Jenkins remotely through RESTful API via an HTTP destination.

The following figure 6.2 shows a prototype example of the proposed SAPUI5 application:

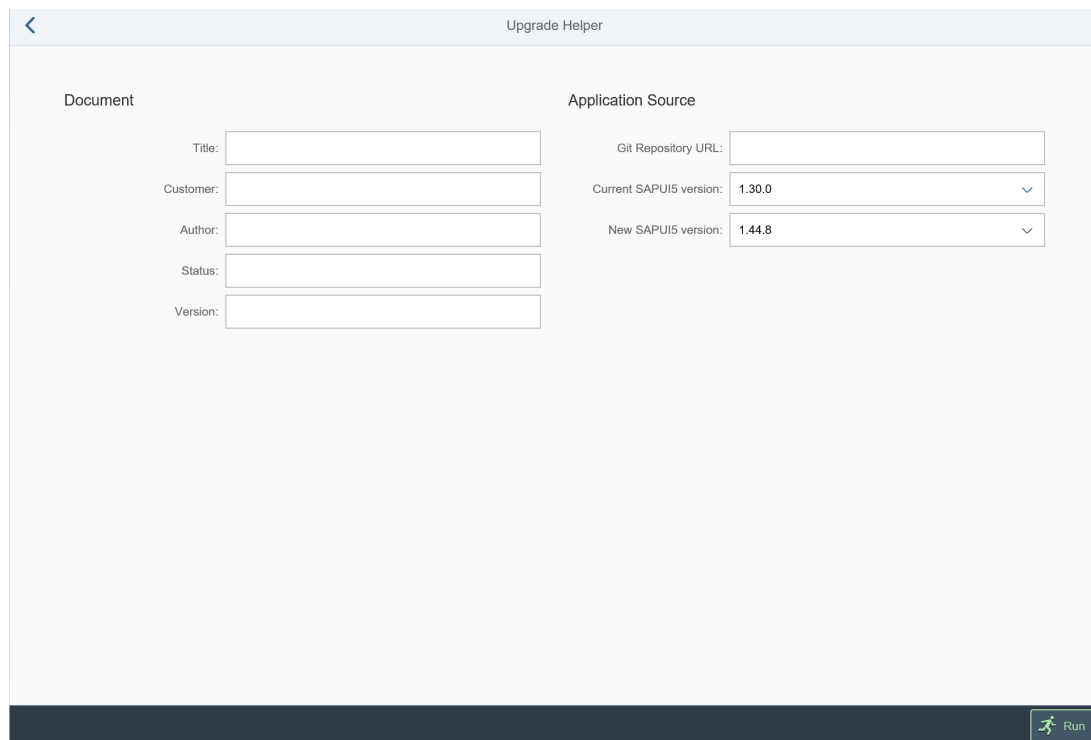


Figure 6.2: Prototype example of a UI of the SAPUI5 application

6.3.5 Automatic indexing of the SAPUI5 library

Currently indexing and downloading the SAPUI5 library is done manually, as mentioned earlier, by using the **bower** command. This solution can be enhanced by automatically downloading a new version of the SAPUI5 library, whenever it is released. This can be achieved by configuring an additional Jenkins job. This job can be configured to take as a parameter the SAPUI5 version to be indexed and to execute first “bower” with the given version, followed by triggering the Node.js module “libStoring” for maintaining the library version on SAP HANA database.

Appendix

Appendix A

npm Modules used

```
"dependencies": {  
  "app-root-path": "^2.0.1",  
  "archy": "1.0.0",  
  "body-parser": "1.17.0",  
  "bower": "1.8.2",  
  "common-substrings": "1.0.2",  
  "count-files": "^2.6.2",  
  "diff": "3.3.0",  
  "directory-tree": "1.2.0",  
  "docxtemplater": "3.0.5",  
  "dotenv": "4.0.0",  
  "escape-string-regexp": "1.0.5",  
  "eslint": "3.17.1",  
  "express": "4.15.0",  
  "get-line-from-pos": "^1.0.0",  
  "git-user-name": "1.2.0",  
  "grunt": "^0.4.5",  
  "grunt-cli": "1.2.0",  
  "grunt-contrib-nodeunit": "~0.2.0",  
  "grunt-execute": "0.2.2",  
  "grunt-open": "0.2.3",  
  "grunt-openui5": "^0.12.0",  
  "hdb": "0.12.1",  
  "http-proxy": "1.16.2",  
  "jsdoc": "3.4.3",  
  "jszip": "2.6.1",  
  "line-count": "0.1.0",  
  "linenumber": "1.0.0",  
  "list-directory-contents": "0.0.3",  
  "make-runnable": "1.3.6",  
  "multiline": "^1.0.2",  
  "node-async-loop": "1.2.2",  
  "parse-comments": "0.4.3",  
  "path": "0.12.7",  
  "properties-parser": "0.3.1",  
  "randomstring": "1.1.5",  
  "run-func": "^1.0.2",  
  "rx-text-search": "1.0.0",  
  "scopenodes": "1.0.0",  
  "simple-git": "1.67.0",  
  "string-search": "1.2.0",  
  "strip-comments": "0.4.4",  
  "traverse": "0.6.6",  
  "xml-parser": "1.2.1"  
},
```

Figure A.1: package.json file with the used npm modules

Appendix B

Upgrade Helper Output Document

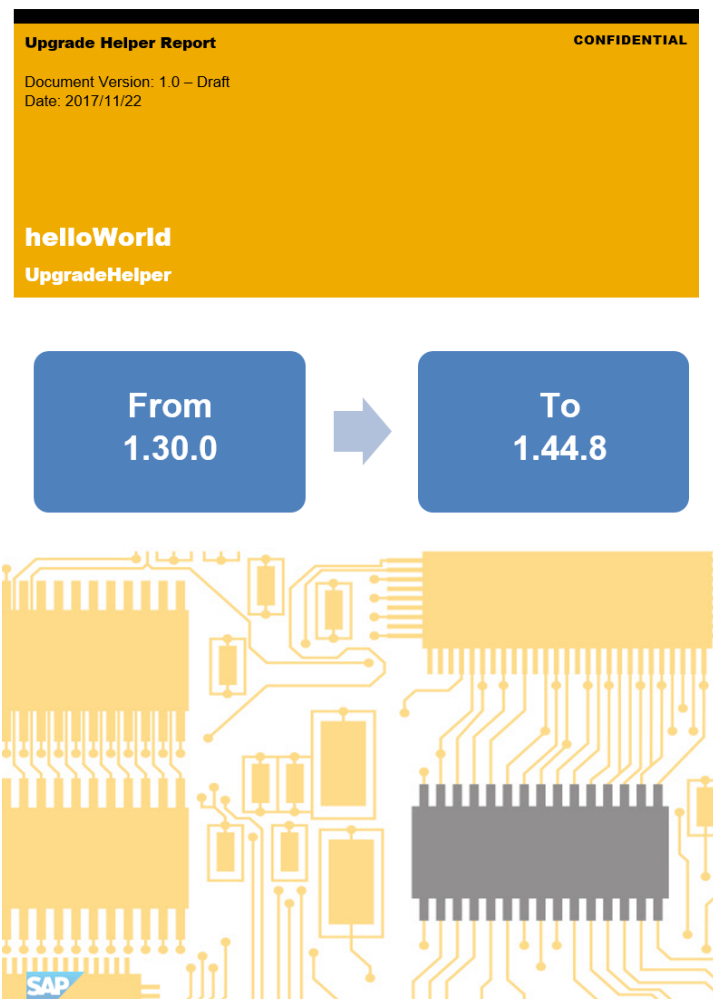


Figure B.1: Cover page of the document

Table of Contents

1	General	3
2	Upgrade Impact Analysis.....	4
3	Deprecated Standard Controls	7
3.1	sap.ui.commons.Label.....	7
4	Deprecated Properties of Standard Controls.....	8
4.1	selectOnFocus.....	8
5	Analysis of used Standard Controls	9

Figure B.2: Table of contents

1 General

Namespace

sap.ui.demo.wt

SAPUI5 Version

1.30.0

OData Service

Name: invoiceRemote

URL: /destinations/northwind/V2/Northwind/Northwind.svc/

Figure B.3: General information about the SAPUI5 app

2 Upgrade Impact Analysis


Control / Class	Modified by	Occurrence	 Risk Factor
sap.ui.core.mvc.View	50.20%	5 time(s)	2509.78
sap.ui.core.mvc.XMLView	118.53%	2 time(s)	2370.61
sap.m.Button	44.78%	5 time(s)	2239.22
sap.ui.core.mvc.Controller	52.31%	4 time(s)	2092.31
sap.ui.model.json.JSONModel	44.10%	3 time(s)	1322.92
sap.m.Page	64.29%	2 time(s)	1285.71
sap.m.Text	14.43%	8 time(s)	1154.43
sap.m.ObjectAttribute	44.27%	2 time(s)	885.42
sap.ui.test.Opa5	81.47%	1 time(s)	814.71
sap.m.ColumnListItem	76.26%	1 time(s)	762.59
sap.m.Dialog	74.05%	1 time(s)	740.45
sap.m.SearchField	68.63%	1 time(s)	686.27
sap.ui.model.Filter	54.33%	1 time(s)	543.31
sap.m.RatingIndicator	47.29%	1 time(s)	472.92
sap.m.Panel	46.63%	1 time(s)	466.26
sap.m.Column	9.17%	5 time(s)	458.65

Figure B.4: Upgrade Impact Analysis list

sap.ui.core.Icon	38.72%	1 time(s)	387.23
sap.m.ObjectNumber	19.23%	2 time(s)	384.62
sap.m.Table	34.48%	1 time(s)	344.77
sap.ui.core.Control	33.33%	1 time(s)	333.33
sap.m.ObjectHeader	33.33%	1 time(s)	333.33
sap.m.Input	33.11%	1 time(s)	331.05
sap.m.MessageToast	11.53%	2 time(s)	230.60
sap.ui.core.util.MockServer	22.65%	1 time(s)	226.47
sap.ui.core.UIComponent	22.40%	1 time(s)	224.03
sap.m.ObjectIdentifier	17.59%	1 time(s)	175.95
sap.m.Title	14.50%	1 time(s)	145.04
sap.ui.commons.Label	12.31%	1 time(s)	123.08
sap.ui.model.resource.ResourceModel	12.28%	1 time(s)	122.81
sap.ui.model.FilterOperator	5.68%	1 time(s)	56.82
sap.m.Toolbar	4.64%	1 time(s)	46.41
sap.m.Label	3.57%	1 time(s)	35.71
sap.m.App	2.54%	1 time(s)	25.38
sap.ui.core.routing.History	0.00%	1 time(s)	0.00
sap.m.ToolbarSpacer	0.00%	1 time(s)	0.00

Figure B.5: Continue: Upgrade Impact Analysis list

3 Deprecated Standard Controls

3.1 sap.ui.commons.Label

File Name

HelloPanel.view.xml

File Path

/webapp/view/HelloPanel.view.xml

Code Snippet**Line Number 13**

```
<commons:Label
```

Deprecated Since

Since version 1.38. Instead, use the sap.m.Label control.

Figure B.6: List of deprecated controls

4 Deprecated Properties of Standard Controls

4.1 selectOnFocus

Control Name

sap.m.SearchField

File Name

InvoiceList.view.xml

File Path

/webapp/view/InvoiceList.view.xml

Code Snippet**Line Number 20**

```
<SearchField width="50%" search="onFilterInvoices"
selectOnFocus="false"/>
```

Deprecated Since

Since version 1.38. This parameter is deprecated and has no effect in run time. The cursor position of a focused search field is restored after re-rendering automatically.

Figure B.7: List of deprecated properties

5 Analysis of used Standard Controls

Control Name	# times used	Locations
sap.m.Text	8	<ul style="list-style-type: none"> HelloPanel.view.xml 1 times InvoiceList.view.xml 7 times
sap.ui.core.mvc.View	5	<ul style="list-style-type: none"> App.view.xml 1 times Detail.view.xml 1 times HelloPanel.view.xml 1 times InvoiceList.view.xml 1 times Overview.view.xml 1 times
sap.m.Button	5	<ul style="list-style-type: none"> HelloPanel.view.xml 2 times Overview.view.xml 1 times HelloDialog.fragment.xml 1 times ProductRating.js 1 times
sap.m.Column	5	<ul style="list-style-type: none"> InvoiceList.view.xml 5 times
sap.ui.core.mvc.Controller	4	<ul style="list-style-type: none"> App.controller.js 1 times Detail.controller.js 1 times HelloPanel.controller.js 1 times InvoiceList.controller.js 1 times
sap.ui.model.json.JSONModel	3	<ul style="list-style-type: none"> Component.js 1 times Detail.controller.js 1 times InvoiceList.controller.js 1 times
sap.m.Page	2	<ul style="list-style-type: none"> Detail.view.xml 1 times Overview.view.xml 1 times
sap.ui.core.mvc.XMLView	2	<ul style="list-style-type: none"> Overview.view.xml 2 times
sap.m.ObjectAttribute	2	<ul style="list-style-type: none"> Detail.view.xml 2 times
sap.ui.demo.wt.model.formatter	2	<ul style="list-style-type: none"> InvoiceList.controller.js 1 times formatter.js 1 times
sap.m.MessageToast	2	<ul style="list-style-type: none"> Detail.controller.js 1 times HelloPanel.controller.js 1 times
sap.m.ObjectNumber	2	<ul style="list-style-type: none"> InvoiceList.view.xml 2 times

Figure B.8: Analysis of the used controls within the app

sap.m.Toolbar	1	<ul style="list-style-type: none"> InvoiceList.view.xml 1 times
sap.m.Title	1	<ul style="list-style-type: none"> InvoiceList.view.xml 1 times
sap.m.ToolbarSpacer	1	<ul style="list-style-type: none"> InvoiceList.view.xml 1 times
sap.m.SearchField	1	<ul style="list-style-type: none"> InvoiceList.view.xml 1 times
sap.ui.demo.wt.control.ProductRating	1	<ul style="list-style-type: none"> Detail.view.xml 1 times
sap.m.ColumnListItem	1	<ul style="list-style-type: none"> InvoiceList.view.xml 1 times
sap.m.Panel	1	<ul style="list-style-type: none"> HelloPanel.view.xml 1 times
sap.m.ObjectIdentifier	1	<ul style="list-style-type: none"> InvoiceList.view.xml 1 times
sap.ui.commons.Label	1	<ul style="list-style-type: none"> HelloPanel.view.xml 1 times
sap.ui.core.FragmentDefinition	1	<ul style="list-style-type: none"> HelloDialog.fragment.xml 1 times
sap.m.App	1	<ul style="list-style-type: none"> App.view.xml 1 times
sap.ui.core.Icon	1	<ul style="list-style-type: none"> HelloDialog.fragment.xml 1 times
sap.ui.Device	1	<ul style="list-style-type: none"> Component.js 1 times
sap.ui.core.UIComponent	1	<ul style="list-style-type: none"> Component.js 1 times
sap.ui.demo.wt.controller.HelloDialog	1	<ul style="list-style-type: none"> Component.js 1 times
sap.m.ObjectHeader	1	<ul style="list-style-type: none"> Detail.view.xml 1 times
sap.m.Label	1	<ul style="list-style-type: none"> ProductRating.js 1 times
sap.m.RatingIndicator	1	<ul style="list-style-type: none"> ProductRating.js 1 times

Figure B.9: Continue: Analysis of the used controls within the app

sap.ui.core.Control	1	<ul style="list-style-type: none"> ProductRating.js 1 times
sap.m.Input	1	<ul style="list-style-type: none"> HelloPanel.view.xml 1 times
sap.ui.thirdparty.sinon-qunit	1	<ul style="list-style-type: none"> formatter.js 1 times
sap.ui.core.routing.History	1	<ul style="list-style-type: none"> Detail.controller.js 1 times
sap.ui.base.Object	1	<ul style="list-style-type: none"> HelloDialog.js 1 times
sap.m.Table	1	<ul style="list-style-type: none"> InvoiceList.view.xml 1 times
sap.ui.model.Filter	1	<ul style="list-style-type: none"> InvoiceList.controller.js 1 times
sap.ui.model.FilterOperator	1	<ul style="list-style-type: none"> InvoiceList.controller.js 1 times
sap.ui.core.util.Mock Server	1	<ul style="list-style-type: none"> mockserver.js 1 times
sap.ui.test.opaQunit	1	<ul style="list-style-type: none"> navigationJourney.js 1 times
sap.ui.test.Opa5	1	<ul style="list-style-type: none"> App.js 1 times
sap.ui.model.resource.ResourceModel	1	<ul style="list-style-type: none"> formatter.js 1 times
sap.ui.thirdparty.sinon	1	<ul style="list-style-type: none"> formatter.js 1 times
sap.m.Dialog	1	<ul style="list-style-type: none"> HelloDialog.fragment.xml 1 times

Figure B.10: Continue: Analysis of the used controls within the app

Bibliography

- [1] ESLint. <http://eslint.org/>. Last accessed 21.07.2017.
- [2] Git. <https://git-scm.com>. Last accessed 24.07.2017.
- [3] Jenkins. <https://jenkins.io/>. Last accessed 26.07.2017.
- [4] JSHint. <http://jshint.com/>. Last accessed 21.07.2017.
- [5] JSLint. <http://www.jshint.com/>. Last accessed 21.07.2017.
- [6] Nexus Repository Sonatype. <https://books.sonatype.com/nexus-book/pdf3/nxbook-pdf.pdf>. Last accessed 04.08.2017.
- [7] Node.js. <https://www.nodejs.org>. Last accessed 22.07.2017.
- [8] Npm. <https://www.npmjs.com/>. Last accessed 22.07.2017.
- [9] parse-comments module. <https://www.npmjs.com/package/parse-comments>. Last accessed 18.09.2017.
- [10] SAP HANA Developer Guide. <https://help.sap.com/viewer/52715f71adba4aaeb480d946c742d1f6/2.0.01/en-US/627f113fa17d481cab2347248012acb1.html>. Last accessed 27.07.2017.
- [11] SAPUI5. <https://www.sap.com/developer/topics/ui5.html>. Last accessed 25.07.2017.
- [12] SAPUI5 Developer Guide. https://help.sap.com/erp_hcm_ias2_2015_02/helpdata/en/95/d113be50ae40d5b0b562b84d715227/frameset.htm. Last accessed 25.07.2017.
- [13] SAPUI5 Development Toolkit Documentation. https://help.sap.com/saphelp_nw74/helpdata/en/91/f079dc6f4d1014b6dd926db0e91070/frameset.htm. Last accessed 29.07.2017.

- [14] SAPUI5 Development Toolkit Documentation 2. https://help.sap.com/saphelp_uiaddon20/helpdata/en/a8/7ca843bcee469f82a9072927a7dcdb/frameset.htm. Last accessed 01.08.2017.
- [15] SAPUI5: UI Development Toolkit for HTML5. https://help.sap.com/http.svc/rc/625468c53c7044a08531cb6b01bb325a/External/en-US/SAPUI5_en.pdf. Last accessed 25.07.2017.
- [16] SAPUI5: UI Development Toolkit for HTML5. https://help.sap.com/saphelp_uiaddon20/helpdata/en/91/f346786f4d1014b6dd926db0e91070/frameset.htm. Last accessed 12.10.2017.
- [17] SAPUI5: UI Development Toolkit for HTML5: Component.js file. https://help.sap.com/saphelp_nw74/helpdata/en/27/ce0e4987cd426f8fa3e60836316428/frameset.htm. Last accessed 19.10.2017.
- [18] SAPUI5: UI Development Toolkit for HTML5: neo-app.json File. https://help.sap.com/saphelp_uiaddon20/helpdata/en/28/fa7538c67e4280a0b7708de2951278/frameset.htm. Last accessed 19.10.2017.
- [19] SAPUI5: UI Development Toolkit for HTML5: SAPUI5 Folder Structure. <https://help.sap.com/viewer/0ce0b8c56fa74dd897ffffda8407e8272/7.5.6/en-US/003f755d46d34dd1bbce9ffe08c8d46a.html>. Last accessed 19.10.2017.
- [20] scopenodes module. <https://www.npmjs.com/package/scopenodes>. Last accessed 18.09.2017.
- [21] Nathaniel Ayewah, David Hovemeyer, J David Morgenthaler, John Penix, and William Pugh. Using static analysis to find bugs. *IEEE software*, 25(5), 2008.
- [22] Thoms Ball. The concept of dynamic analysis. In *ACM SIGSOFT Software Engineering Notes*, volume 24, pages 216–234. Springer-Verlag, 1999.
- [23] Ulrich Bayer, Andreas Moser, Christopher Kruegel, and Engin Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, 2006.
- [24] Martin Bodin, Arthur Charguéraud, Daniele Filaretti, Philippa Gardner, Sergio Maffeis, Daiva Naudziuniene, Alan Schmitt, and Gareth Smith. A trusted mechanised JavaScript specification. In *ACM SIGPLAN Notices*, volume 49, pages 87–100. ACM, 2014.
- [25] Jamie Cool, Bradley Abrams, and Eric Zinda. System and method for automatically upgrading a software application, July 11 2002. US Patent App. 10/195,132.
- [26] Frank Dabek, Nikolai Zeldovich, Frans Kaashoek, David Mazières, and Robert Morris. Event-driven programming for robust software. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 186–189. ACM, 2002.

- [27] Michael D Ernst. Static and dynamic analysis: Synergy and duality. In *WODA 2003: ICSE Workshop on Dynamic Analysis*, pages 24–27, 2003.
- [28] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. SAP HANA database: data management for modern business applications. *ACM Sigmod Record*, 40(4):45–51, 2012.
- [29] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The SAP HANA Database—An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.
- [30] Amin Milani Fard and Ali Mesbah. JSNOSE: Detecting JavaScript code smells. In *Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on*, pages 116–125. IEEE, 2013.
- [31] Benjamin Harrison, Daniel Nussbaum, and David Kranz. System and methods for providing compatibility across multiple versions of a software system, December 14 2000. US Patent App. 09/736,949.
- [32] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Pixy: A static analysis tool for detecting web application vulnerabilities. In *Security and Privacy, 2006 IEEE Symposium on*, pages 6–pp. IEEE, 2006.
- [33] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.
- [34] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC’01. Proceedings. Fifth IEEE International*, pages 118–127. IEEE, 2001.
- [35] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development.* ” O’Reilly Media, Inc.”, 2012.
- [36] Gregor Richards, Sylvain Lebresne, Brian Burg, and Jan Vitek. An analysis of the dynamic behavior of JavaScript programs. In *ACM Sigplan Notices*, volume 45, pages 1–12. ACM, 2010.
- [37] Vishal Sikka, Franz Färber, Wolfgang Lehner, Sang Kyun Cha, Thomas Peh, and Christof Bornhövd. Efficient transaction processing in SAP HANA database: the end of a column store myth. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 731–742. ACM, 2012.
- [38] Richard A Stupek Jr, David S Shaffer, Curtis R Jones, Steve Davis, and William D Justice Jr. Automatic Computer Upgrading, December 17 1996. US Patent 5,586,304.

- [39] Stefan Tilkov and Steve Vinoski. Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, 2010.
- [40] Tajkia Rahman Toma and Md Shariful Islam. An efficient mechanism of generating call graph for JavaScript using dynamic analysis in web application. In *Informatics, Electronics & Vision (ICIEV), 2014 International Conference on*, pages 1–6. IEEE, 2014.