

Abstract

In this thesis an implementation of an n=gram based language model using the Modified Kneser=Ney smoothing algorithm with the open source Apache Spark large scale data processing engine and the open source document oriented database MongoDB will be presented. This language model will then be used to generate sentences from input sets of keywords.

Contents

1	Introduction	3
2	Fundamentals and State of The Art	4
2.1	State of the Art	4
2.1.1	TODO some papers	4
2.2	Interaction Diagrams	4
2.3	Scenario Testing	4
2.4	Behavior-Driven Development	5
2.4.1	the other one(forgot name)	5
2.4.2	Gherkin Language	5
2.5	Model-View-ViewModel	5
2.5.1	View	6
2.5.2	View-Model	6
2.5.3	Model	6
2.6	Vue.js	6
2.6.1	Components	6
2.6.2	Reactivity	6
2.6.3	Directives	6
2.6.4	Data Binding	7
2.6.5	Vue.js directives	7
2.6.6	Structure of a Component	7
2.7	ESLint	7
2.7.1	Architecture	7
2.8	Rules	8
2.8.1	AST Explorer	9
2.8.2	ESTree AST	9
2.8.3	ESLint Parser Vue AST	9
A	Appendix	10

Chapter 1

Introduction

Chapter 2

Fundamentals and State of The Art

2.1 State of the Art

TODO

2.1.1 TODO some papers

2.2 Interaction Diagrams

2.3 Scenario Testing

Scenario testing, was originally introduced in Kaner [Kan03] and later as Kaner [Kan13]. The author defines scenarios as hypothetical stories, which aid a person in understanding a complex system or problem. Scenario tests are tests, which are based on such scenarios. [Kan13, p. 1] Further, [Kan03, pp. 2–5] defines five characteristics, which make up a good scenario test as follows: A Scenario test must be

- based on a story - based on a description of how the program is being used
- motivating - stakeholders have interest in this test succeeding and would see to it's resolution
- credible - probable to happen in the real world
- complex - complex use, data or environment
- easy to evaluate - it should be easy to tell if the test succeeded or failed based on the results

Kaner [Kan13] describes the biggest advantages of scenario testing to be - understanding and learning the product in early stages of development(1), connecting of testing and requirement documentations(2), exposing shortcomings in delivering of desired benefits(3), exploration of expert use of the program(4), expose requirement related issues(5).

2.4 Behavior-Driven Development

Behavior-Driven Development (BDD), pioneered by North [Nor06] is a software development process, that combines principles from Test-Driven Development and Domain-Driven design [EE04].

Its main goal is to specify a system in terms of its functionality (i.e. its behaviors) with a simple domain-specific language (DSL) making use of English-like sentences. This stimulates collaboration between developers and non-technical stakeholders and further results in a closer connection between acceptance criteria for a given function and matching tests used for its validation.

BDD splits a user story into multiple scenarios, each formulated in the form of *Given*, *When*, *Then* statements, respectively specifying the prerequisite/context, event and outcomes of a scenario.

[TODO] example here

At present ... there based on the division of behavior descriptions and behaviors. Such as Jest/Jasmine combine behavior descriptions and behaviors into one, whereas as Cucumber uses a DSL named Gherkin to specify the behavior descriptions and provides a set of tools to generate behaviors.

2.4.1 the other one(forgot name)

splitting into ...

2.4.2 Gherkin Language

2.5 Model-View-ViewModel

Model-View-ViewModel (MVVM) is a design pattern, which helps in creating a clear separation between business and presentation logic and User interface (UI) of an application. [Bri17, pp. 7–9]

In MVVM there are three core components - the view, model and view model. Those components are clearly separated from each other - the view is aware of the view model and the view model is aware of the model. However, this does not hold in reverse - the model is unaware of the view model and the view model is unaware of the view.

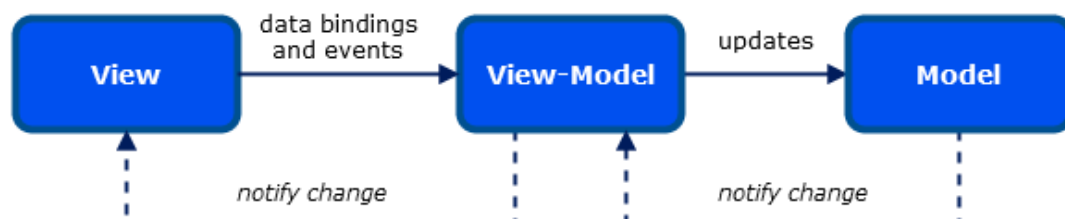


Figure 2.1: MVVM design pattern overview, adapted from [Bri17, p. 7]

2.5.1 View

The view is what the user sees. It is responsible for the structure, layout and appearance of the application.

2.5.2 View-Model

The view model implements event handlers and properties, to which the view can bind to. It also notifies the view of any changes to the underlying data. It defines the functionality, offered by the UI, but the view determines how it is presented.

2.5.3 Model

The model encapsulates the data of the application and validation its logic.

2.6 Vue.js

Vue.js [vue21a] is a progressive front end framework for building user interfaces and single-page applications based on the MVVM design pattern described in 2.5 [Mac18] [21a].

2.6.1 Components

At the core of Vue.js are components, which are small, self-contained, composable and often reusable custom elements. Almost any type of application can be represented as a tree of components [21a].

In more concrete terms, a Vue.js component is a single file with the extension of *.vue*, which consists of a *template*, *script* and optional *style* part. The *template* is a HTML-based template, which can be parsed specification compliant browsers and HTML parsers. It can contain other components or html elements and is equivalent to the *view* in MVVM.

The *code* section of a Vue.js component includes the view-model of the component. It has a special json object *data*, which is equivalent to the MVVM model. The *script* part of a computed includes css-like styles.

data binding is a general technique that binds data sources from the provider and consumer together and synchronizes them.

2.6.2 Reactivity

...enables data binding

2.6.3 Directives

Vue.js enables one way bindings(from source - data to target - component or html tag) via the *v-bind* (line X,Y) or *moustache* syntax (line Z). Bindings can contain expressions (line X,Y).

Two way binding can be achieved using the *v-model* directive (line,X,Y,Z). Event handlers can be bound by the method name or also expressions.

2.6.4 Data Binding

Vue.js provides support for various forms of Data Binding via a special syntax. Both the data and computed objects of a *Vue.js* component are reactive

Via a special syntax *Vue.js* - one way - two way - event bindings - inline expressions - computed properties

2.6.5 Vue.js directives

2.6.6 Structure of a Component

(template, code etc.)

bindings two way, one way

g - data - computed properties

template part code part bindings

2.7 ESLint

ESLint [21b] is a linting tool (linter) for ECMAScript/JavaScript. Linters are static code analysis tool, which can be used to flag and potentially automatically fix common code issues and enforce consistent code styling.

2.7.1 Architecture

At a very high level, ESLint consists of

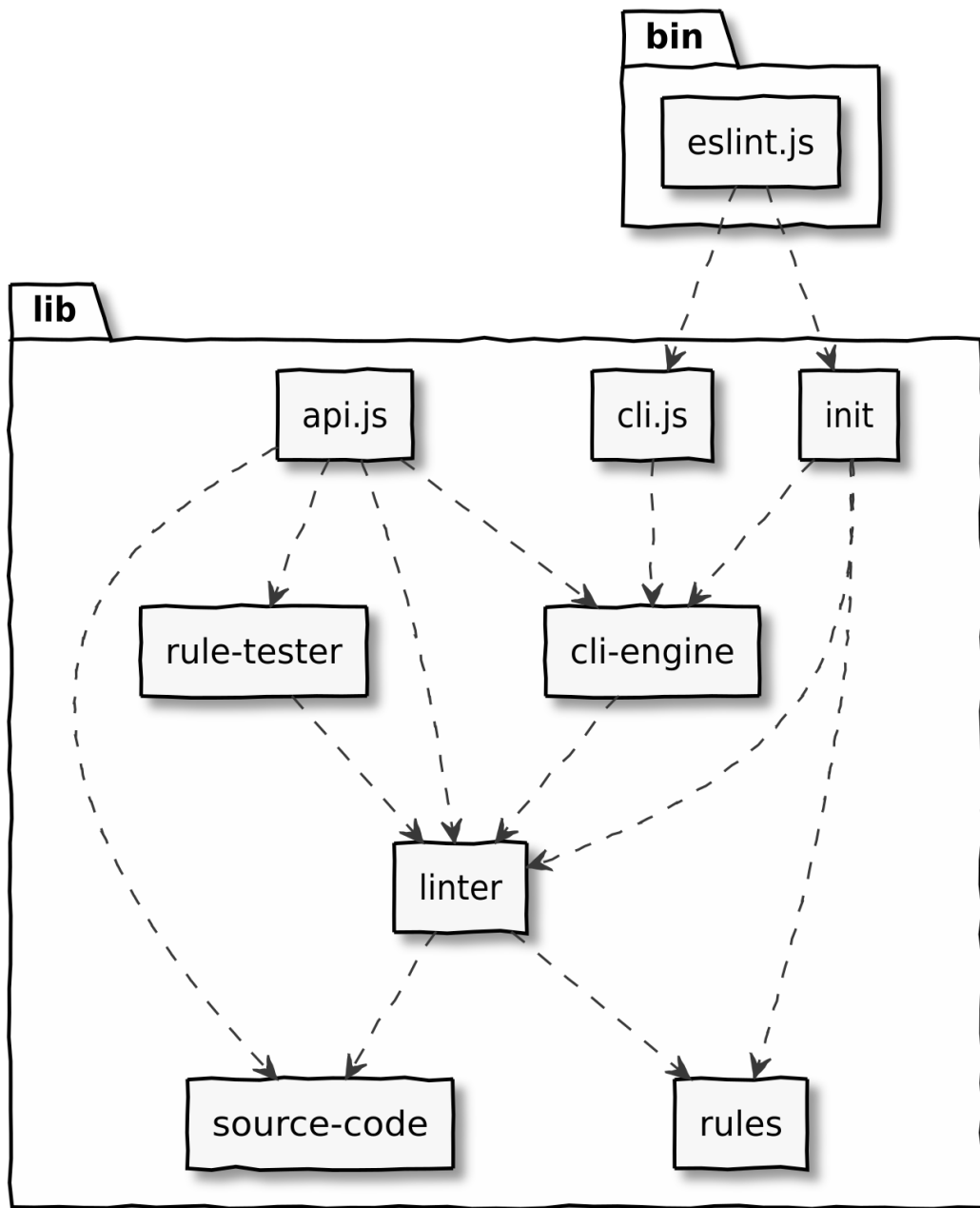


Figure 2.2: ESLint Architecture taken from [21c]

2.8 Rules

At the core of ESLint are rules. Rules are extensible pieces of code, bundled as plugins, which can be used to verify various aspects of code. An example would be a rule, which checks for matching closing paranthesis. Each rule consists of a *metadata* object and a *create* function. The metadata object includes metadata such as documentation strings, the type of the rule and whehter it is fixable or not.

Based on type, rules can be either *suggestions*, *problems* or *layout*. Suggestions indicate some

improvement, but are not required and would not cause the linting to fail. Problems on the other hand would result in a linting failure. Layouts are rules that care mainly about the formatting of code, such as whitespaces, semicolons, etc.

If the *fixable* property is specified, it indicates that the errors reported by this rule can be automatically fixed. This can be applied via the `--fix` command line option. It has two possible values - *code* or *whitespace* indicating the type of fixes, that this rule would apply. For example in Integrated development environments (IDEs) fixable *code* errors would show a fix shortcut displayed next to them and *whitespace* rules could be applied when saving the file.

The *create* function of rules takes as arguments a *context* and returns an object of methods which are called by ESLint for each node based on the Visitor pattern while traversing the Abstract syntax tree (AST). ESLint provides a very powerful matching mechanism for specifying what nodes to match called selectors [21d] inspired by estools [est21a].

TODO custom architecture image

TODO what are selectors

TODO what does context provide?

2.8.1 AST Explorer

An incredibly useful tool when working with ASTs is AST Explorer, developed by Kling [Kli21]. It enables the exploration of syntax tree generated by various parsers and also includes the vue-eslint-parser [vue21b]

2.8.2 ESTree AST

By default ESLint uses the [esl21] parser to parse JavaScript source code into an AST as defined by ESTree specification [est21b]. When Parsing *.vue* files ESLint uses this parser for the code inside the *<script>* tag.

2.8.3 ESLint Parser Vue AST

In order to parse the *<template>* section of *.vue* files, ESLint uses the vue-eslint-parser [vue21b]. This parser outputs an AST compliant with their own ASTspecification, defined in [vue21c].

So what if there is text here?

Appendix A

Appendix

List of Figures

2.1	MVVM design pattern overview, adapted from [Bri17, p. 7]	5
2.2	ESLint Architecture taken from [21c]	8

List of Tables

Bibliography

- [Kan03] Cem Kaner. “The Power of ‘What If... and Nine Ways to Fuel Your Imagination: Cem Kaner on Scenario Testing”. In: *Software Testing and Quality Engineering* 5 (2003), pp. 16–22.
- [Kan13] Cem Kaner. “An introduction to scenario testing”. In: *Florida Institute of Technology, Melbourne* (2013), pp. 1–13.
- [Nor06] Dan North. “Behavior modification”. In: *Better Software* 8.3 (2006), p. 26. URL: <https://dannorth.net/introducing-bdd/>.
- [EE04] Eric J Evans and Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.
- [Bri17] David Britch. *Enterprise Application Patterns using Xamarin. Forms*. 2017. URL: <https://github.com/dotnet-architecture/eBooks/blob/master/current/xamarin-forms/Enterprise-Application-Patterns-using-XamarinForms.pdf>.
- [vue21a] vuejs. *vue*. [Online; accessed 29. Jan. 2021]. Jan. 2021. URL: <https://github.com/vuejs/vue>.
- [Mac18] Callum Macrae. *Vue.js: Up and Running: Building Accessible and Performant Web Apps*. " O'Reilly Media, Inc.", 2018.
- [21a] *Introduction Vue.js*. [Online; accessed 29. Jan. 2021]. Jan. 2021. URL: <https://vuejs.org/v2/guide>.
- [21b] *ESLint - Pluggable JavaScript linter*. [Online; accessed 25. Jan. 2021]. Jan. 2021. URL: <https://eslint.org>.
- [21c] *Architecture*. [Online; accessed 27. Jan. 2021]. Jan. 2021. URL: <https://eslint.org/docs/developer-guide/architecture>.
- [21d] *Selectors*. [Online; accessed 27. Jan. 2021]. Jan. 2021. URL: <https://eslint.org/docs/developer-guide/selectors>.
- [est21a] estools. *Esquery*. [Online; accessed 27. Jan. 2021]. Jan. 2021. URL: <https://github.com/estools/esquery>.

- [Kli21] Felix Kling. *AST Explorer*. [Online; accessed 27. Jan. 2021]. Jan. 2021. URL: <https://github.com/fkling/astexplorer>.
- [vue21b] vuejs. *vue-eslint-parser*. [Online; accessed 27. Jan. 2021]. Jan. 2021. URL: <https://github.com/vuejs/vue-eslint-parser#readme>.
- [esl21] eslint. *Espree*. [Online; accessed 25. Jan. 2021]. Jan. 2021. URL: <https://github.com/eslint/espree>.
- [est21b] estree. *estree*. [Online; accessed 27. Jan. 2021]. Jan. 2021. URL: <https://github.com/estree/estree>.
- [vue21c] vuejs. *vue-eslint-parser*. [Online; accessed 27. Jan. 2021]. Jan. 2021. URL: <https://github.com/vuejs/vue-eslint-parser/blob/master/docs/ast.md>.