Hackathon 2015

SLATE: Slide-Making in the Web-Age

Author:
Ritu Kundu

Developer:
Ritu Kundu

A tutorial for beginners

March 2015

Abstract

SLATE: Slide-Making in the Web-Age

by Ritu KUNDU

A new software tool for making and delivering slide-based presentations has been implemented, keeping in mind the existing tools in popular use today, their advantages and drawbacks. The tool, called *SLATE*, takes input in form of a LaTeX like language which completely specifies the presentation, generates HTML/CSS/JS as output which can be rendered and run on any common web-browser. The in-built language has been developed to be syntactically similar to the LaTeX language and especially the BEAMER class. SLATE provides a lot of novel features which are anticipated to be of quite useful for authors of presentations, especially those within academia already familiar with LaTeX. ...

Contents

Abstract i

List of Figures iv

List of Tables v

Abbreviations vi

1 Introduction 1
- 1.1 Objectives . . . 1
  - 1.1.1 Primary Objectives . . . 2
    - 1.1.1.1 Secondary Objectives . . . 2
- 1.2 Roadmap . . . 3

2 Technical Background 4
- 2.1 LaTeX and Beamer . . . 4
- 2.2 Elm . . . 5
  - 2.2.1 Why Elm? . . . 5
- 2.3 PEG . . . 6
  - 2.3.1 Why PEG.js . . . 7

3 System Design 8
- 3.1 The *SLATE* Language . . . 8
  - 3.1.1 Is it really a new language? . . . 8
  - 3.1.2 Language Description . . . 9
  - 3.1.3 Language Syntax . . . 13
- 3.2 Features . . . 16
  - 3.2.1 Authoring features: . . . 17
    - 3.2.1.1 Theme: . . . 17
    - 3.2.1.2 Interactive Content: Quiz . . . 18
    - 3.2.1.3 Smart-Display . . . 18
    - 3.2.1.4 Animation . . . 18
  - 3.2.2 Delivery Features . . . 21
    - 3.2.2.1 Navigational Menu bar . . . 21
    - 3.2.2.2 Zoom mode . . . 21
    - 3.2.2.3 Scribble mode . . . 21
- 3.3 Architecture . . . 22

4 Conclusion **23**
4.1 Future Extensions 23
4.2 Known Issues 24
4.3 Contributions 24

A Slate: Theme and Navigation **25**
A.1 Different Layout themes 25
A.2 Different Navigation Menu-Bar Formats 26

B Slate Language **28**
B.1 Appearance 29
B.1.1 Theme 29
B.1.2 Style 30
B.1.3 Layout 36
B.2 Information 37
B.3 Content 38
B.3.1 Slide 38
B.3.2 Supportbin 42
B.3.3 Quiz 42

C A sample *SLATE* file **43**

Bibliography **47**

List of Figures

2.1 A Sample Beamer Frame 5

3.1 Slide Components and Elements 11
3.2 A sample SLATE slide 13
3.3 Checkbox List 18
3.4 Conical Circle List 18
3.5 Container List 19
3.6 Framed List 19
3.7 Full Circle List 19
3.8 Half Circle List 19
3.9 Paragraph List 20
3.10 Plus Equal (Horizontal) 20
3.11 Plus Equal (Vertical) 20
3.12 Plus Minus 20
3.13 *SLATE* Design 22

A.1 Default Layout Theme 26
A.2 InfoLines Layout Theme 26
A.3 Miniframes(No Foot) Layout Theme 26
A.4 Sidebar(left) Layout Theme 26
A.5 Split Layout Theme 27
A.6 Tree SLayout Theme 27

B.1 Sober Style Theme 29
B.2 Striking Style Theme 30

List of Tables

3.1 Inner-Elements 10
3.2 Outer-Elements 12

Abbreviations

ADT	**A**bstract **D**ata **T**ypes
CSS	**C**ascading **S**tyle **S**heets
DSDM	**D**ynamic **S**ystems **D**evelopment **M**ethod
FRP	**F**unctional **R**eactive **P**rogramming
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**yper**T**ext **M**arkup **L**anguage
WYSIWYG	**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et

Chapter 1

Introduction

Slide-based presentations are today a very popular means of communicating ideas and information, especially in business and educational settings. Traditionally, most such presentations are prepared and delivered using desktop software (predominantly Microsoft Word), but increasingly, software tools that enable users to create and/or deliver slides over the World-wide-web are gaining in usage.

Most slide-making tools are based on the What-you-see-is-what-you-get (WYSIWYG) paradigm of user interaction, where the user interacts with the software through a GUI and builds/manipulates the slides and content directly. While this mode of interaction is easy to learn and has therefore become very popular among users, there are inherent problems with it. An alternative is to let the user create the slides using code, much as LaTeX [1] allows one to create documents.

The aim of this project was to create a browser based slide-making and presenting tool aimed at users who might already be familiar with tools such as LaTeX and prefer them over WYSIWYG tools. Presented here is *SLATE*, a browser based application that lets the user create slide presentations by coding in a LaTeX -like language. Besides this primary objective, the software also has features which enhance both the creation and presentation of slide-shows.

1.1 Objectives

The objectives of the project can be classified as primary objectives and secondary objectives based on their relative importance. Primary goals are deemed as absolutely essential for the project and secondary ones are the desirable features for raising quality of the tool produced and making it more usable.

1.1.1 Primary Objectives

Input:

As already mentioned, the tool is aimed at catering to the needs of academia, and LaTeX is the most widely used language for documentation. Input should be coded in a language which is similar to Latex, thus resulting in following benefits:

1. Learning a new language poses a certain psychological difficulty and demands time-investments. Ensuring the language to be similar to LaTeX eases out these issues.
2. Same content that was written for other documents(reports, papers etc.) can be reused with little changes, if any.

Output:

Output should be able to be rendered in a web browser because of its suitability to handle dynamic content and support for developing highly responsive applications. Web browsers also make an application portable and accessible on different platforms and devices.

1.1.1.1 Secondary Objectives

SLATE aims to incorporate the following features.

1. Separation of 'Appearance' from the 'Content' Visual appearance should be separated from content specifications so that *author* can concentrate on actual content while authoring presentation rather than how that content should look
2. Provision for Rich Animations: Animations help to retain concepts and processes as they help in developing a mental model of how these processes work [2]. Animation plays an important role in dispensing information effectively. Ease of building even an exceedingly complex animation and its smooth rendering should be facilitated.
3. Provision for smart display of data formulated as lists: "Smart-Art" of Microsoft Powerpoint displays the data given as lists in a graphical format which is both visually more appealing and make more sense by graphically illustrating the relationships between various segments of the data. A feature like smart-art is aimed to be included in this tool.
4. Provision for Interactive Content like Quizzes: Quizzes can be used to intrigue the audience and thus enhance interaction. Hence, there should be a provision for the user to add questions, along with correct answers, while creating a presentation, which can be used later when delivering the presentation. When the audience's response is fed, it gets compared to the correct answer and results should be shown accordingly.

5. Provision for breaking linear/sequential flow: To break the imposition of following linear flow of navigation through slides, 'out-of-order' navigation at desired points can be provided. It will also solve the problem of referring back to the already displayed slides.

6. Provision for Additional "Back-up" Information: There should be a provision to provide some additional information that might be required to explain something or answer a question that could be asked by audience. This information is not intended to be displayed by *presenter* under 'normal circumstances'. It should not hinder the normal flow of presentation. The presenter should be able to switch to this "back-up" information, on a need-basis.

7. Provision for "On-Demand" Zooming: Provision to zoom an element(text, figure, image etc) on the current slide being presented, would grant the choice to have a closer look when needed. *Author* should not be expected to anticipate where zooming might be required and prepare accordingly.

8. Provision for "Scribbling": Content to be included in the presentation gets locked in the authoring stage itself. But, in many situations, like explaining some concept or to draw attention to a particular part of slide, presenter feels a need to write/draw on the slide. This spontaneity and flexibility to stretch beyond prepared content can be achieved by allowing for a /textitscribble mode where presenter is given a free hand to draw.

1.2 Roadmap

Rest of the document is structured as follows:

Chapter 2 provides a technical background which will give a better understanding of the project, including a brief introduction to LaTeX and Beamer along with an overview of the programming language Elm and the rationale behind choosing it to write the core of *SLATE*.

Chapter 3 describes the design aspects of the project. It introduces the *SLATE* language, specifies the features of the *SLATE* tool and discusses the software architecture.

Chapter 4 discusses the possible extensions, known issues and the contribution of the tool.

Chapter 2

Technical Background

For its overall design, and especially for the language design, *SLATE* owes a lot to LaTeX and Beamer. This chapter begins with an introduction to these software packages. It then introduces Elm, a new functional reactive programming language, in which the core of *SLATE* has been implemented. Having a basic understanding of Elm will enable one to better understand the implementation details. It also explains the rationale behind choosing Elm and other software tools as well.

2.1 LaTeX and Beamer

LaTeX is a document creation system including a mark-up language. It is based on Donald E. Knuth's TeX typesetting language and is widely used in academia. Conceptually, LaTeX is similar to a mark-up language like HTML but the difference is that HTML is a functional mark-up language whereas LaTeX is designed as page layout language [3]. Since *SLATE* aims to cater to the same user base, the embedded language within *SLATE* is designed to be similar in syntax to to facilitate learning for new users.

Beamer is a LaTeX class which can be used to create slides for presentations. From a user's point of view, creating a presentation in Beamer is much like writing a document with LaTeX. A Beamer presentation input file much like has a preamble and a body. The Body contains sections and subsections, which further contain series of frames. Each frame is a series of slides. A frame can be built into a particular 'environment' which are similar to those provided in LaTeX. Additionally, Beamer provides numerous themes to change the appearance of the presentation as required. It also allows the user to change the layout, font or color globally at one level and also provides the ability to change finer details at another level. Dynamic effects are simulated in Beamer by using 'overlays'. An overlay is used to support incremental display of information and is a way to mimic animation.

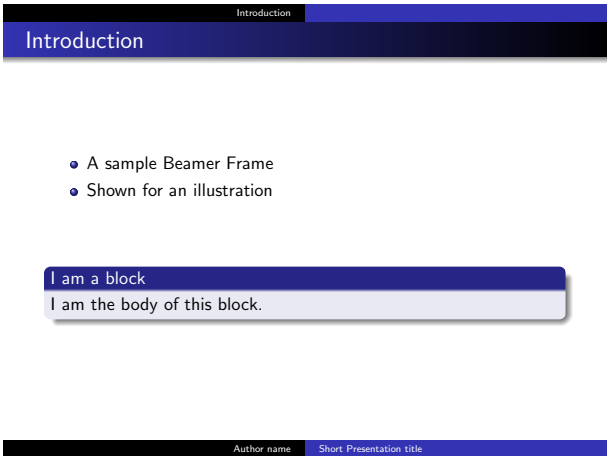

FIGURE 2.1: A Sample Beamer Frame

2.2 Elm

Elm's website describes it as A functional reactive language for interactive applications. The Elm compiler takes in valid Elm code and compiles it down to a combination of HTML, CSS and JS code, which can then be run/rendered on any standards-compliant browser. Functional Reactive Programming is a programming paradigm which lets the programmer deal with events, providing control-flow structures for them. This is an alternative paradigm to the event-driven nature inherent in JavaScript.

The following bit of code can serve as an example of the brevity and power of Elm for coding browser applications.

```
import Mouse
main = lift asText Mouse.position
```

This is a complete Elm program that prints and updates the co-ordinates of the mouse continually. Mouse.position is a built-in Signal whose value is the co-ordinates of the mouse pointer at any given time, whereas asText is a function that takes a value and renders it as text (analogous to the print I/O function in most languages). The lift function enables a static function (which takes as input a value rather than a time-varying Signal) to take a Signal and produce a Signal as output.

2.2.1 Why Elm?

Since *SLATE* is meant to be rendered and run on a browser, the choice is between using pure JavaScript or one of the many languages which compile down to JavaScript (like Elm). Although a compile-to-JavaScript language adds one extra step of compilation and thus overall

complexity, the significant reduction in the size and complexity of the actual code makes it a good engineering choice. Some of the ways in which code-complexity is reduced are:

1. Elm is a much less verbose language than JavaScript, leading to much fewer Lines of Code (LOC) overall. Significantly, coding in Elm means not having to separately write HTML, CSS and JS code.
2. The build-in data-structures in Elm (like dictionaries, lists and records) score over JavaScript data structures like mutable Arrays and Objects, making for cleaner, less buggy code.
3. The functional nature of Elm and the Strong Static Type system also encourages clean, maintainable code.
4. Most importantly, Elm provides excellent support for interactivity, with native support for streams of events (known as Signals) which proves to be very useful for an application which aims to provide excellent interactivity support like *SLATE*.

2.3 PEG

A parsing expression grammar, or PEG, is a type of analytic formal grammar which is closely related to the family of top-down parsing languages [4]. Syntactically, PEGs are similar to context free grammars (CFGs) but due to different interpretation of choice operator, PEGs are not ambiguous like CFGs. In other words, a parsed string will always have a unique parse tree in PEGs unlike CFGs. This is because choice operator is ordered in PEGs and thus selects the first match while it is ambiguous in CFGs. For example, consider a grammar rule like

```
if_statement := "if" "(" expression ")" statement "else" statement
              | "if" "(" expression ")" statement;
```

For an input

```
if (a) if (b) x else y
```

CFG can parse it as

```
if_statement(a, if_statement(b, x, y))
or
if_statement(a, if_statement(b, x), y)
```

whereas PEG will choose the first rule and would produce only a single parse tree.

PEGs vave many advantages over CFGs such as

- Because of the ordered choice operator, PEGs are free from ambiguity.
- *Packrat* implementation of PEGs uses *memoization* and parses PEGs in linear time
- PEGs don't require *tokenization* as a separate step. Token rules can be written just like any grammar rule

However, PEGs also have some down-sides like more memory consumption and less expressive power.

2.3.1 Why PEG.js

SLATE is required to have parser to parse the *SLATE* language. Elm made a good choice for the implementation of *SLATE*, but it did not have any parsing support at the time the choice was made. Therefore a parser was needed to be written in some other language. JavaScript makes a natural choice for a web-based application. A parser generator was chosen rather than writing the parser 'by hand' to save time so that other important and more complicated aspects of the project could be focused on.
PEG.js takes as input a set of parsing rules and automatically generates the requisite parser tool encoded in JavaScript. This generated parser can take as input grammatically-valid*SLATE*code and produce a parse-tree.

Chapter 3

System Design

This chapter begins with an introduction to the *SLATE* language. It gives a brief description of how a presentation-document is structured and defined in the *SLATE* language. The syntax and the semantics of the language are presented in an informal style giving examples for clarity.

This chapter also highlights the features provided by *SLATE* in the *authoring* and *presenting* stages. The architecture of the tool is presented at the end of this chapter along with the design choices while also analysing other possible designs.

As a side-note, a convention for has been followed to italicize a word denoting some new concept or feature of the tool when it is introduced. Later, that word is typed as in normal style with first letter capitalized.

3.1 The *SLATE* Language

SLATE, just like LaTeX, has a document mark-up language whose goal is to enable the creation of presentation-documents. The *author* specifies the content with commands for how that content must be rendered on screen while presenting the document.

3.1.1 Is it really a new language?

As described earlier, *SLATE* derives heavily from the vocabulary of Beamer besides structuring the document more or less in the same way as in Beamer. This is because of the following reasons:

- The target users of *SLATE* would normally use Beamer, so using the same keywords can facilitate them to switch easily between Beamer and *SLATE*. Keeping the same keywords

alleviates the need for them to remember new keywords and saves them time and effort. Additionally, a simple *copy and paste* of content might work (with appropriate editing) in case the *author* wishes to make a *SLATE* document from a previously written document in Beamer.

- A document in *SLATE* is structured as consisting of a series of *frames* where a *frame* can be seen as being composed of various components like *Headline, footline* etc. which has a natural parallel to how a human mind would break up the slide visually.

A natural question that comes to mind is why should *SLATE* be considered a new language in that case. Or, why was a new language even needed when one could use the Beamer language and transpile it to be rendered in a web-browser? In fact, the Beamer and *SLATE* languages differ in many respects. First, the way the *appearance* of a presentation is expressed and handled in *SLATE* is very different from how it is done in Beamer. *SLATE* has tried to come up with a more obvious and natural way to express the look and feel of the presentation. Second, *SLATE* is aimed at producing a browser-displayable presentation that gives more scope for dynamicity and interactivity. Therefore, a language describing such a document must have the capability to encode dynamic and interactive content. Also, Beamer has a large set of commands that can be seen as superfluous for *SLATE*. For example *SLATE* provides a "Zoom on-demand" feature. It frees up the author from the need to anticipate where zooming-in might be required and having to build that in. Thus *SLATE* doesn't need to deal with commands to enable this zoom-effect.

3.1.2 Language Description

The *SLATE* language structures a presentation as a series of slides. This sequence of slides can logically be divided into *sections, subsections or sub-subsections.* A slide fills up the screen when rendered in web-browser. The following components make up a slide on the screen:

1. *Headline:* This component can serve to provide a navigation menu-bar or meta-information such as the names of sections/subsections to which the current slide belongs; author, title, subtitle of the presentation, slide number etc. It is the top-most component which is *slide-width* wide. Currently, its height is fixed at 10% of the *slide-height.*

2. *Footline:* Same as the Headline with the only difference that it is the bottom-most component.

3. *Sidebars:* There can be a sidebar on either the left or right ends or on both ends of the slide. A sidebar vertically stretches between the bottom of the headline and the top of the Footline. The default width of the sidebar is 20% of the slide-width which can not be changed by the *author* currently. It serves the same purpose as the Headline.

4. *Slide Title:* This component is to display the title of the slide. It is placed immediately below the headline and flows between sidebars.

5. *Slide Subtitle:* It is the component containing the subtitle of the slide and is placed immediately below the slide title.

6. *Symbol Panel:* It is the component containing *navigational symbols* and *mode-change symbols*. *Navigational symbols* provide the ability to move to the next or previous slide/-section/subsection or to move back or forth on the *animation-timeline* of the same slide. *animation-timeline* has been explained later in the next section.

   Mode-change symbols are there for switching between *modes*. texitModes are explained later in Chapter **??** while describing the implementation. This component is placed above the Footline and between the Sidebars.

7. *Content:* The component containing the actual contents of the slide. It expands between Sidebars horizontally and between the Subtitle and Symbol panels vertically.

These components can be classified as *Outer* and *Inner* depending on the purpose they serve. *Outer* components are Headline, Footline, Sidebars, Symbol panel, Slide title and Slide subtitle. And Content makes up the *Inner*class.*Outer* components provide the meta-information whereas *Inner* component contain the actual content of the slide. This classification helps in setting the appearance of presentation as seen later. Each of the components contain *elements*. An Element is basically a piece/block of information. For example a block carrying name of the author of presentation is an *Author-element* and an image in the content of the slide is an *Image-element*.

Table 3.1 and 3.2 describe each element that can be there in a slide and figure 3.1 gives the elements that can be placed in each of the components.

Element	*Description*
Simple Text	Textbox containing simple text
Block	A block of text with a title
Alerted Block	A block with alerted text in title
Framebox	Text enclosed in a frame
Imagebox	Graphics enclosed in a box
Itemize	A bulleted List
Enumerate	Numbered List
Description	A definition list

Table 3.1: Inner-Elements

Moreover, there can be *marker slides* inserted in the presentation at each logical division point. A marker slide can be of the following types:

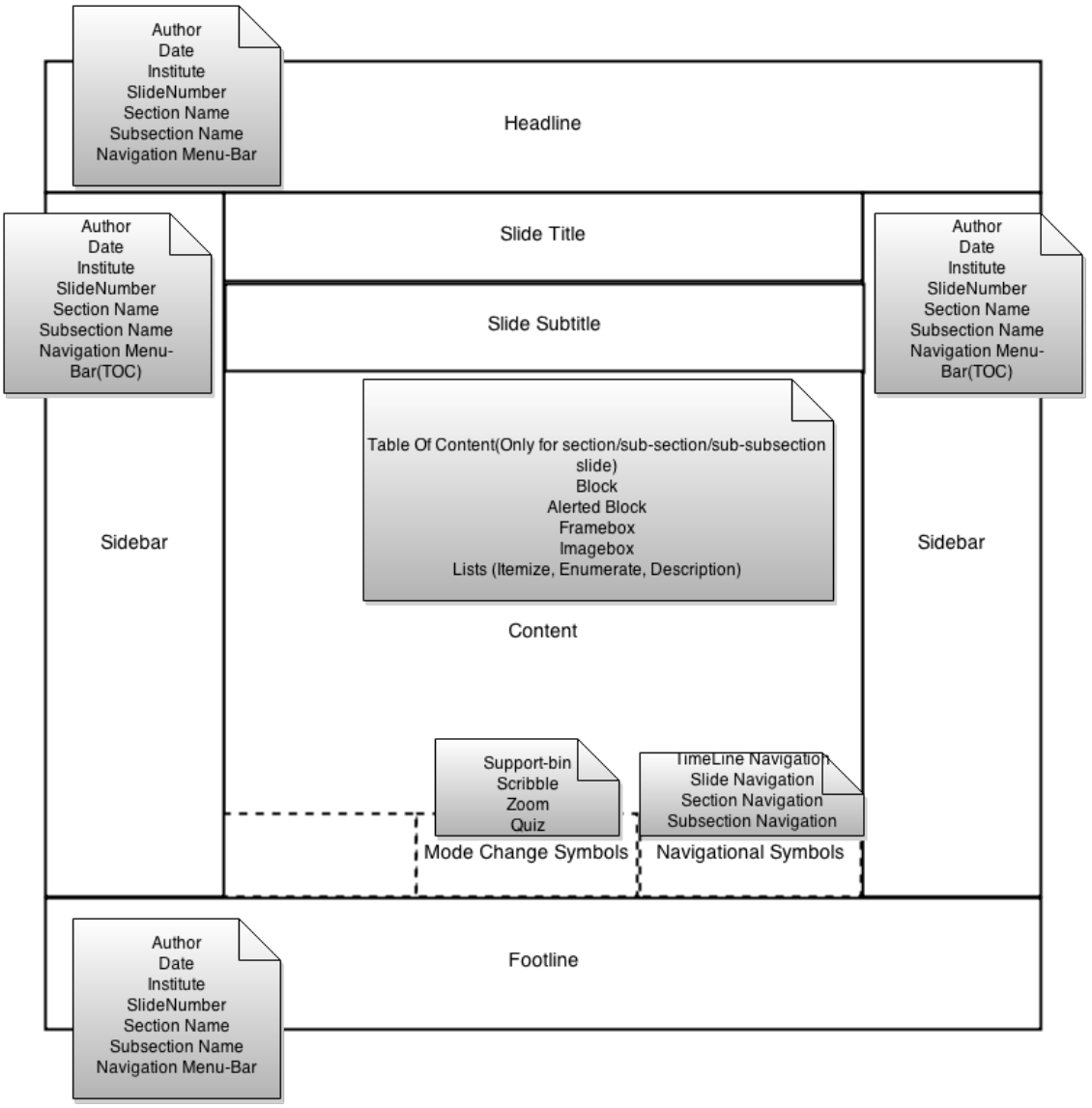

FIGURE 3.1: Slide Components and Elements

1. *Title Slide:* Slide marking the beginning of the presentation. It can contain elements such as Title, Subtitle, Author, Institute, Date and a picture (Title graphic)

2. *Table of Content Slide:* Slide immediately after the Title slide and contains table of content

3. *Section Slide:* Slide marking the beginning of sections. It can contain Section, Logo, Table of Contents and Section Graphic

4. *Subsection Slide:* Slide marking the beginning of subsections. It can contain Section, Subsection, Logo, Table of Contents and Section Graphic

Element	*Description*
Title	Title of the presentation
Subtitle	Subtitle of the presentation
Author	Contains the name of the authors of the presentation
Institute	Contains the institutes of the authors of the presentation
Date	Contains the date of the presentation
Logo	Logo Graphic for the presentation
Slide Title	Title of the slide
Slide Subtitle	NSubtitle of the slide
SlideNumber	Number of the current slide
SlideNumberAndTotal	Number of the current slide and total number of the slides
Section	Name of the section to which current slide belongs
Subsection	Name of the section to which current slide belongs
Title Graphic	Graphic on the Title Slide
Section Graphic	Graphic on the Section Slide
Subsection Graphic	Graphic on the Subsection Slide
Section	Name of the section to which current slide belongs
TableOfContent	A hierarchical-tree representing the relationships between sections, subsections and subsubsections
Navigation-Menu Bars	These are explained later in section describing Features
Navigational symbols	symbols for navigation: slidenavigation, sectionnavigation, subsectionnavigation, backforward
Mode-change symbols	Explained later in Chapter **??**

Table 3.2: Outer-Elements

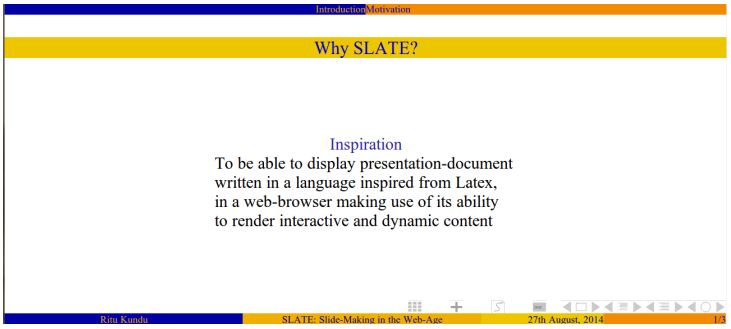

FIGURE 3.2: A sample SLATE slide

3.1.3 Language Syntax

A sample document in *SLATE* language would look like the following:

```
\begin{presentation}
\begin{appearance}
\usestyletheme{striking}
\end{appearance}

\begin{information}
 \title{SLATE: Slide-Making in the Web-Age}
 \authors{Ritu Kundu}
 \date{27th August, 2014}
\end{information}

\begin{content}
 \section{Introduction}
  \subsection{Motivation}
   \begin{slide}{Why SLATE?}
    \begin{block}
     \title{Inspiration}
      To be able to display presentation-document written in a language inspired from
        Latex, in a web-browser \\making use of its ability to render interactive
                  and dynamic content
    \end{block}
   \end{slide}
\end{content}

\end{presentation}
```

It will produce ta slide as shown in figure 3.2.

In general a container starts with **\begin{**⟨*container-name* ⟩**}** and ends with **\end{**⟨*container-name* ⟩**}**. The outermost container is 'presentation'. It has three parts:

```
\begin{presentation}
 \begin{appearance}
  ...
 \end{appearance}
 \begin{information}
  ...
 \end{information}
 \begin{content}
  ...
 \end{content}
\end{presentation}
```

- *Appearance Part*: It holds the specifications pertaining to appearance of the presentation.
- *Information Part*: It holds the meta information about the presentation like title, subtitle, author etc.
- *Content Part*: It holds the content of the presentation in the form of slides

Presented here is a very brief description of a subset of commands of *SLATE* language. Complete detailed documentation of the language is in appendix B and can be better understood after going through this chapter in full.

1. Appearance: Appearance of a presentation can be changed by changing the 'style' of an element or the 'layout' of a component. These concepts are explained later in the next section 3.2 where the 'theme' feature has been explained. Here, only the syntax of commands to change the appearance is introduced.

   - Style of an element can be changed by commands

     \setcanvas{⟨*element/component* ⟩**}**[⟨*canvas-parameters* ⟩]
     \setcolor{⟨*element/component* ⟩**}**[⟨*color-parameters* ⟩]
     \setfont{⟨*element/component* ⟩**}**[⟨*font-parameters* ⟩]

     \setcanvas sets the canvas border and padding of an element. Container of the element can be specified following the **in** keyword. **\setcolor** sets the colors of various attributes like background(bg), font color(fg), color of border and color of

padding. These will be ignored if border or padding is set to 'none'. **\setfont** sets the font-seriesof the element. An illustrative usage of the commands is as follows:

```
\setcanvas{author in headline}[border=thin!dashed, padding=0.1\%]
\setcolor{author in headline}[bg=white, fg=red, border=black, padding=none]
\setfont{author in headline}["times new roman", "monotype"]
```

- The layout of a component can be changed by a command:

  \setlayout{⟨*component* ⟩**}{**⟨*various insert commands* ⟩**}**

  \setlayout specifies the layout: The elements in a component and their arrangement. The elements to be inserted in that component are specified using **\insert** command. It takes the element to be inserted and any associated parameters. Only valid elements can be inserted in a component in accordance with 3.2 The elements are arranged from top to bottom in the order in which they are inserted. To arrange them side by side **\column** command can be used. It divides the component in specified number of columns of specified width. Elements inserted in a column will be stacked top to bottom while various columns will be arranged side by side. Following is an example that places Navigational menu-bar and Title side by side in Headline.

```
\setlayout{headline}
{\begin{columns}
   \begin{column}[width=50\%, align=t]
    \insert{sectionnavigation}[direction=vertical]
   \end{column}
   \begin{column}[width=50\%]
    \insert{title}
     \insert{subsection}
   \end{column}
 \end{columns}
}
```

2. Information: This section contains the commands giving meta-information about the presentation such as title, subtitle, authors of the presentation etc. For example:

```
\title{\textit{SLATE}: Slide-making in the web age }
\athors{Ritu Kundu}
```

3. Content: The sections explained above are 'global' in effect, i.e., the commands in those section pertain to whole presentation. But Content section is where actual content on a slide and its layout are specified which changes on every slide. It can contain following commands:

\begin{slide}...**\end{slide}**
\begin{supportbin}...**\end{supportbin}**
\quiz{slide}...**\end{quiz}**
\section{⟨*section-name* ⟩**}**
\subsection{⟨*subsection-name* ⟩**}**
\subsubection{⟨*subsubsection-name* ⟩**}**

Supportbin and Quiz have been explained in the next section 3.2. Section/Subsection/Subsubection commands are for logical division. Slide container contains various commands to insert and arrange different type of elements in a slide. The types of elements had been given in table 3.1. Following example gives a glimpse of the commands.

```
\begin{Block}
\title{Block Title}
It is the block body
\end{block}
\begin{itemize}
\item I am first item
\item I am second item
\begin{enumerate}
\item I am first item in nested list.
My parent is second item of outer list
\end{enumerate}
\end{itemize}
\begin{imagebox}[width=10\%]
\caption{Image Caption}
\image{"./figures/image1.jpg"}
\end{imagebox}
```

The command names are meant to be self-explanatory. The arrangement can be specified using **\column** commands as explained previously while describing Appearance commands. More details are in appendix AppendixB.

3.2 Features

SLATE tries to provide some interesting and helpful features. These can be classified into two classes based on the stage of the presentation they are used in. *Authoring* features are used while authoring a presentation and *Delivery* features used while presenting, as the names suggest.

3.2.1 Authoring features:

3.2.1.1 Theme:

The theme is responsible for the way things look, i.e., the overall look and feel of the presentation. Appearance of any component is dictated by a combination of two aspects:

1. *Layout of components:* The layout specification of a component defines which elements it will contain and how these are laid out. For example, the layout of the headline can be configured to specify that it should contain a section-navigational menu and a subsection-navigational menu, each taking up half of the space in headline.
2. *Style of elements:* The style controls the the look of an element. For example, details like background color, font-family, font color, border, padding of an element can be specified by the style of that element. Moreover, the appearance of symbols like elements or sub-elements such as marker/projection of a enumerate/itemize list is defined by the style.

The author can control the appearance at three levels:

1. Theme level enables coarse control. Several layout and style themes are in-built in *SLATE*. Choosing a style theme and layout theme frees the author from specifying the appearance of each and every component and element. If none is selected, a default will be used.
2. Element/component level provides fine-grain control and more flexibility. Appearance of each and every element can be changed independently.
3. Style-Controllers: There is a tree-like hierarchy of style-controllers so as to make customization an easy job. Changing the style of a node will push the change to all nodes of the sub-tree rooted at that node. For example, changing the style of item will change the style of all list-items i.e., item of all every kind of lists (enumeration/itemize/description). This style-controller hierarchy is given in Figure **??** where a shaded box represent the controller exposed for the author to change. Unshaded are being used internally by the tool and can not be changed by author.

Currently, three styling themes are provided in *SLATE*, named default, sober and striking. Six layout themes are in-built in *SLATE*: Default, InfoLines, Miniframes, Split, Tree and Sidebar. If none is chosen then the Default layout theme is used. These are explained in appendix A

Support- Bin This feature enables the author to keep additional content to be displayed as and when required. A bin is identified by a label. The author can attach it to any slide. It can contain the same elements as a slide but no arrangement can be specified. In other words elements are just ‘dumped’ into the bin which can be selected to be displayed on screen while presenting.

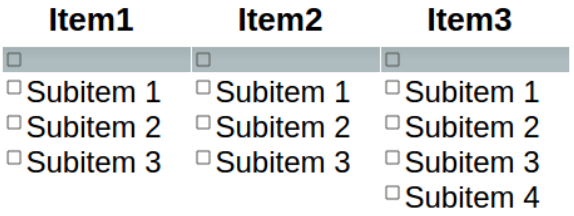

FIGURE 3.3: Checkbox List

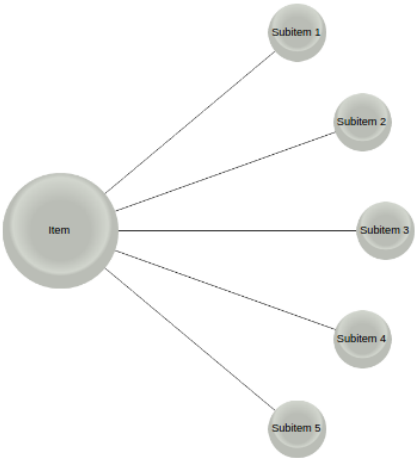

FIGURE 3.4: Conical Circle List

3.2.1.2 Interactive Content: Quiz

Like supportbin, a quiz is identified by a label and can be associated with a slide. It contains elements like *question, correctanswers, answeroptions, explanation.* It can have the following answer types:

1. *Simple:* Answers can be fed into text-boxes.
2. *Multiple Choice:* The answer has to be selected from a drop-down list.

3.2.1.3 Smart-Display

This feature enables the author to present a list into various graphical formats. Various formats of display have been illustrated in figures 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12

3.2.1.4 Animation

Animation support in *SLATE* is based on the concept of a *time-line*, which is composed of *ticks*. A *time-line* begins with the appearance of a slide on the screen. A *tick* is associated with the

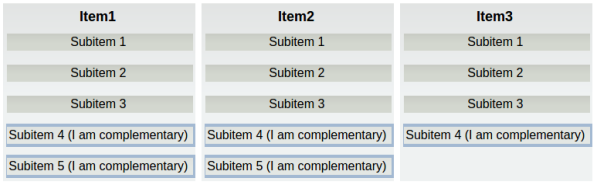

FIGURE 3.5: Container List

FIGURE 3.6: Framed List

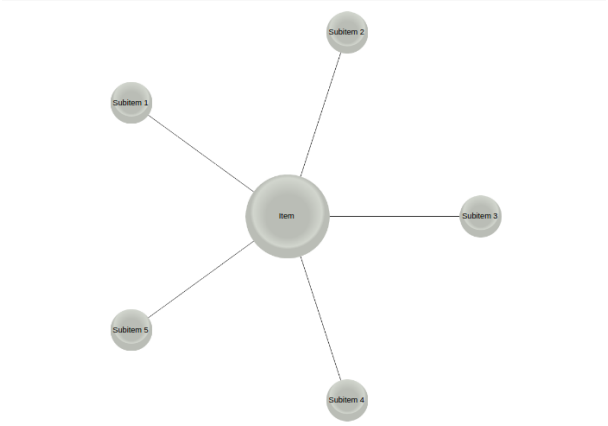

FIGURE 3.7: Full Circle List

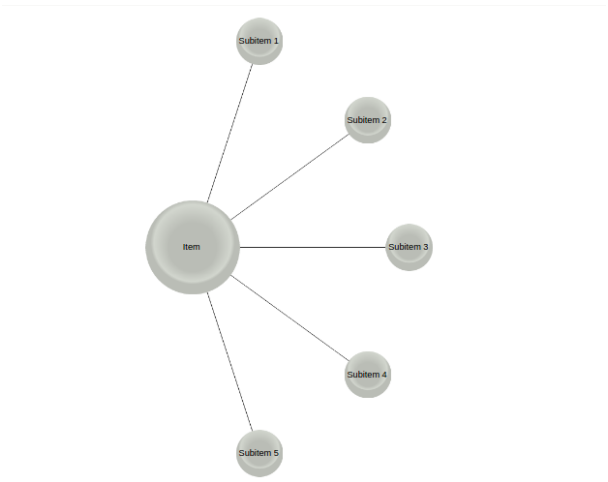

FIGURE 3.8: Half Circle List

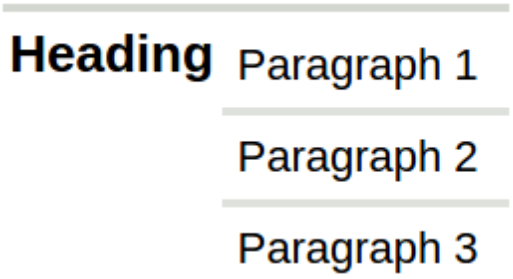

FIGURE 3.9: Paragraph List

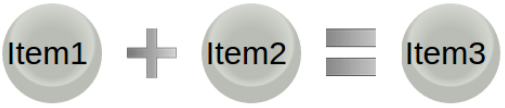

FIGURE 3.10: Plus Equal (Horizontal)

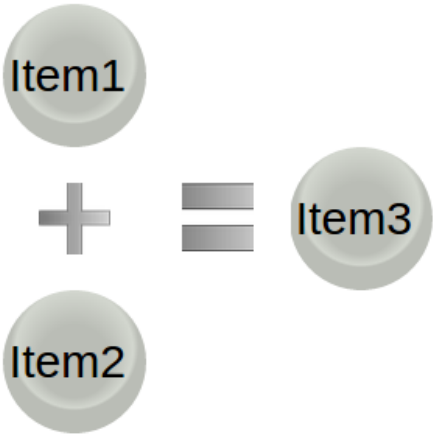

FIGURE 3.11: Plus Equal (Vertical)

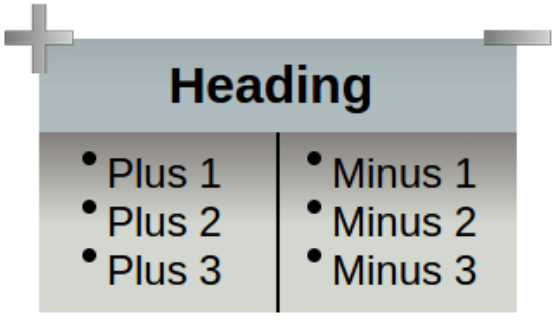

FIGURE 3.12: Plus Minus

entry or exit of some element. The current *tick* will decide which elements will appear on the screen. For example, a block can be specified to appear at *tick* 3 and leave at *tick* 5. This block will the be visible only when the tick is 3, 4 or 5. The backward or foreword move of the current tick can be controlled via external input like mouse clicks and keys presses. An animation can be specified with an element in angle-brackets. For example, the following command will make the block to be visible only at ticks 2,3,7,8,9, and 10

```
\begin{Block}<2,3,7-10>
\title{Block Title
It is the block body
\end{block}
```

Various animation commands are supported in *SLATE* as given in appendix B

3.2.2 Delivery Features

3.2.2.1 Navigational Menu bar

These are the elements representing the logical structure of the presentation, showing sections or sub-sections. The section and sub-section to which the current slide belongs are highlighted. These serve two purposes:

1. Keeping the audience and the presenter oriented by giving a visual context of where current slide fits in.
2. Providing "out-of-order" navigation if needed. Clicking a section will bring the presenter to the first slide of that section.

These can be arranged in various formats:

3.2.2.2 Zoom mode

While presenting, this mode can be entered to zoom any element being displayed on the current slide. That element will be zoomed into to fill up the full screen (keeping the aspect-ratio).

3.2.2.3 Scribble mode

The *Presenter* can scribble on the current slide while in this mode.

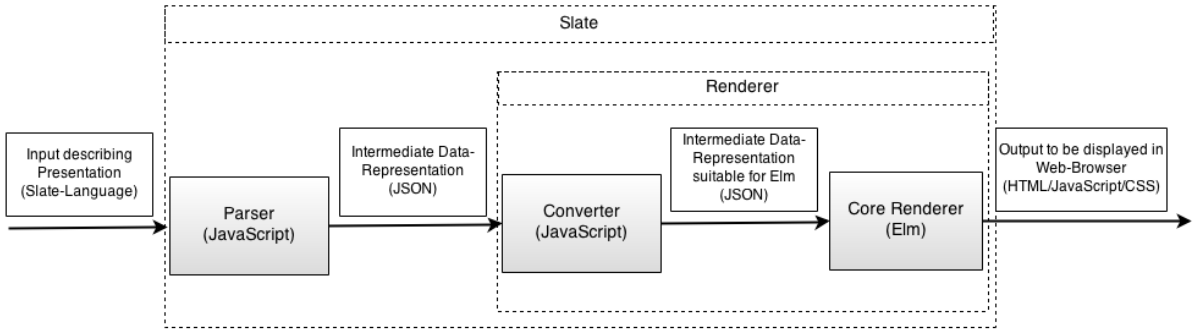

FIGURE 3.13: *SLATE* Design

3.3 Architecture

After thoroughly looking at various feasible design possibilities, it was decided to implement the tool as two separate modules:

- *Parser*
- *Renderer*

The Parser parses the input *SLATE* code and produces an internal intermediate data-structure to be consumed by the Renderer which in-turn produces the output files that can be displayed in a browser. JSON, a lightweight data-interchange format, is used for intermediate data representation for its ease of parsing.

As Elm is a strictly typed language, the Renderer has been implemented as two sub-modules:

- *Converter* that takes the JSON data-structure produced by the Parser and converts it into a JSON representation that is suitable for the Elm code to work on.
- *Core* where the actual rendering process is implemented.

The core Renderer module is implemented as three parts:

1. *Preprocessor:* It converts the data from JSON to an internal representation
2. *Builder:* It builds the 'Elm-elements' that can be displayed in browser, from the internal representation.
3. *Displayer:* It displays the elements depending on the current context. The context changes with key-presses or mouse-clicks.

Chapter 4

Conclusion

4.1 Future Extensions

1. *GUI for naive users :* This tool can be extended to include a GUI for authoring stage so that users who are not familiar with LaTeX can also benefit from the in-presentation features provided by Slate.

2. *Slide transition and animation styles:* Slide transition styles or animation styles like dissolving, rotating etc. can be included in SLATE. It has been observed that the use of slide transitions and animation effects are generally not considered a good presentation style as it distracts the audience from the point being conveyed, however, it can be used to explain the flow of information or simulate an algorithm. It can be implemented using a custom library for Elm called Tweening [http://library.elm-lang.org/catalog/refnil-Tweening/0.1.8]

3. *Presenter View:* SLATE can be extended by adding the provision for the presenter's view where notes added during authoring slides can be displayed to the author but not to the general audience. Further, a clock tracking the time of the presentation and displaying the remaining time can be added in this mode.

4. *Server-hosted SLATE:* Although SLATE doesn't need to be hosted on a server, it can be hosted as an application on a web-server. In this mode of use, new applications can be imagined. One such application can be a hosted quiz: in a class environment for example, students viewing the presentation on their individual devices can answer the hosted quiz and responses sent to a server in real-time which can analyse the responses and store/display statistics.

5. *Inter-operability with LaTeX/BEAMER:* SLATE can be extended by including another module which takes a document written in LaTeX or Beamer and automatically converts that into document in Slate Language that can be rendered in web-browser.

4.2 Known Issues

Listed below are a few of the aspects that *SLATE*lacks at the moment which are seemingly desirable for building a presentation by the target users. These can easily be implemented given the modular design of *SLATE*.

- Some commonly usable environments by LaTeX users, like the Verbatim environment where the text specified will not be interpreted by the parser and will be used as-it-is or the Theorem environment.
- More flexibility over changing the appearance like the ability to define new templates, colours and styles for elements or changing the template of enumerate projection.
- Math mode: Commands to insert mathematical symbols and equations etc.
- Tables: Support for inserting tabular data
- Drawings: Ability to draw diagrams and animate them.
- Comments: There is no provision to write comments while authoring a document.

4.3 Contributions

SLATE is a novel approach to building presentation software as it combines the power of textual input (separation of content and layout, clear specification of the slides) with the versatility of WYSIWYG software (support for animations, dynamic content) and implements it on top of the ubiquitous web-browser, thus making it truly cross platform. A lot of study has also been done to understand the 'feature-landscape' in today's presentation authoring software and many existing and some novel features have been provided in SLATE.

Appendix A

Slate: Theme and Navigation

A.1 Different Layout themes

SLATE provides six different layout themes. They are show in figures A.1,A.2,A.3,A.4,A.5,A.6

- Default Theme: There are no headline or sidebars in this theme. The footline contains only a logo at the bottom-right corner. Title slide contains the title of presentation, followed by subtitle, authors, institute, date, all aligned vertically in the same sequence as mentioned. No Table of Content slide or slides marking the beginning of sections or subsections. All other themes build over the default theme, making required changes as described below.

- InfoLines: Footline contains authors, title, date, slide number total number of slides, spanning 33%, 33%, 17% and 17% of the footline respectively. The Headline is split in equal parts, first displaying section name of the current slide and the second shows the subsection name.

- Miniframes: This theme specifies headline to contain miniframes-navigational menu laid out horizontally along with subsection name below the menu. Several options are given for the footline to choose from, namely author-title, institute-title, author-institiute-title, none.

- Sidebar: It is by far the only theme making use of the sidebars. Sidebar will contain title followed by author followed by a mini toc, arranged vertically. Either of left or right sidebar can be selected to display this meta information.

- Split: This theme splits the headline and footline into two equal parts. Section-navigational menu and subsection-navigational menu are held by headline whereas footline shows name of the section and subsection respectively.

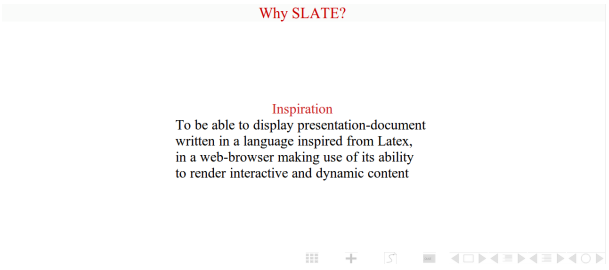

FIGURE A.1: Default Layout Theme

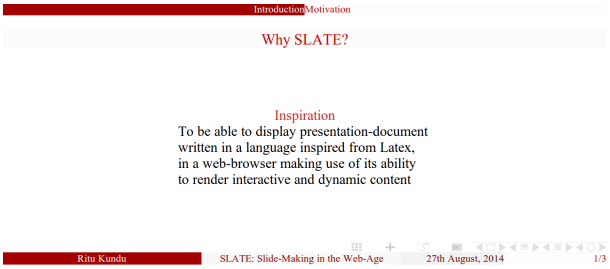

FIGURE A.2: InfoLines Layout Theme

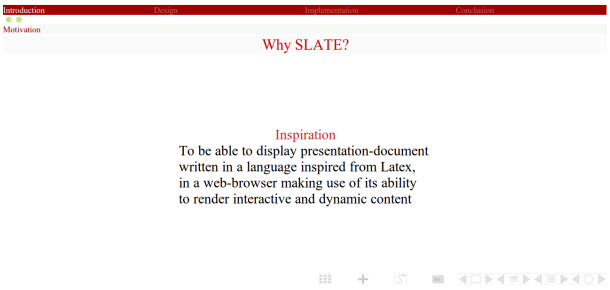

FIGURE A.3: Miniframes(No Foot) Layout Theme

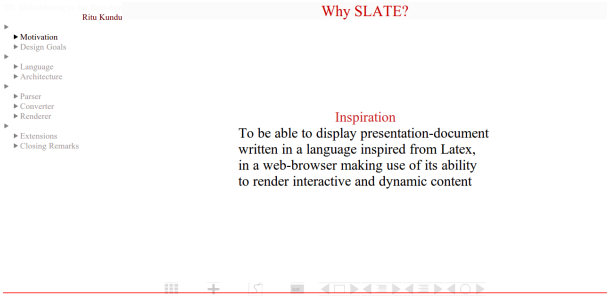

FIGURE A.4: Sidebar(left) Layout Theme

- Tree: It gives title, section-name subsection-name mimicking a tree structure in headline.

A.2 Different Navigation Menu-Bar Formats

- Navigation: A list of sections. Below a section are lists of symbols corresponding to each subsection of that section stacked vertically. This symbol is called a {mini-frame} and corresponds to a sub-sub-section.

FIGURE A.5: Split Layout Theme

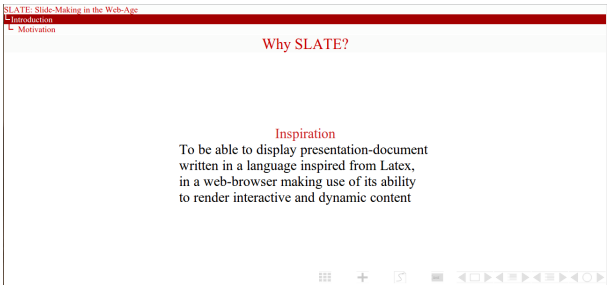

FIGURE A.6: Tree SLayout Theme

- Vertical Navigation : Same as Navigation but the list of sections is placed vertically
- Section Navigation: A list of sections placed horizontally
- Vertical Section Navigation: A list of sections arranged vertically
- Subsection Navigation: A list of sub-sections of the current section placed horizontally
- Vertical Subsection Navigation: A list of sub-sections of the current section stacked vertically

Appendix B

Slate Language

This chapter presents a manual for *SLATE* language. It assumes reader is familiar with the basic LaTeX commands. A small and in-depth introduction to LaTeX can be found at [3] and [5] resp. A convention to color a container name in red, and optional parameters for a command in blue has been adopted. Also as a general note, white space characters between commands are ignored.

A presentation-document specified in *SLATE* language is specified between

```
\begin{presentation}
...
\end{presentation}
```

In general a container is an environment that has a beginning and an end. It starts with **\begin{**⟨*container-name* ⟩**}** and ends with **\end{**⟨*container-name* ⟩**}**.
A document consists of three parts: The outermost container is 'presentation'. It has three parts: A presentation-document specified in *SLATE* language is specified between

```
\begin{appearance}
...
\end{appearance}
\begin{information}
...
\end{information}
 \begin{content}
...
\end{content}
```

- *Appearance Part*: It holds the specifications pertaining to appearance of the presentation. It is an option part.

FIGURE B.1: Sober Style Theme

- *Information Part*: It holds the meta information about the presentation like title, subtitle, author etc. It is an option part.
- *Content Part*: It holds the content of the presentation in the form of slides

B.1 Appearance

Appearance part contains one or more of commands that are used to change appearance of an element in a container or a component or of some elements collectively or of presentation as a whole.

B.1.1 Theme

Slate offers some in-built theme that can be selected and will affect the look of the presentation globally.

Style Themes:
It controls the overall appearance of various elements and components by controlling different attributes like background, foreground, border, padding, size etc. Three style themes have been provided:

- *Default:* It is selected when none is specified by the user.
- *Sober:* It provides a visually 'sober' look.
- *Striking:* It provides a visually colorful look.

The command to select a style theme is as follows:

\usestyletheme{⟨*style theme name* ⟩**}**

For example, **\usestyletheme{**sober**}**and **\usestyletheme{**striking**}**will have a slide look like shown in figures B.1 and B.2 respectively:

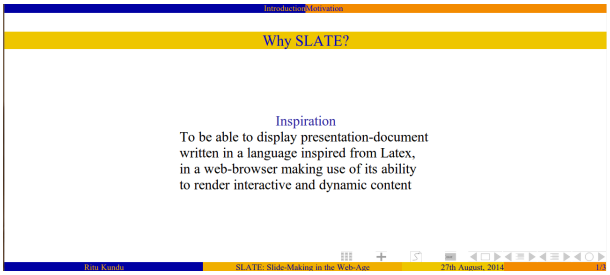

FIGURE B.2: Striking Style Theme

Layout Themes:

It controls the overall arrangement of various elements in outer components. Six style themes are provided that have been explained in appendix A

The command to select a layout theme is as follows:

\uselayouttheme{⟨*layout theme name* ⟩**}**

Layout theme names can be *infolines/ leftsidebar/ rightsidebar/ split/ tree/miniframes:authortitle/miniframes:i*

Font Themes:

It controls the font-series used in the presentation. The command to specify fonts is as follows:

\uselayouttheme{⟨*comma separated font-family* ⟩**}**

There should not be space around commas.

B.1.2 Style

Apart from choosing a style theme, style of an element/component can be customised by the following commands to set canvas,color and font respectively:

\setcanvas{⟨*template name* ⟩**}**[⟨*comma separated canvas options* ⟩][⟨*comma separated additional options* ⟩]

\setcolor{⟨*template name* ⟩**}**[⟨*comma separated color options* ⟩][⟨*comma separated additional options* ⟩]

\setfont{⟨*template name* ⟩**}**[⟨*comma separated font options* ⟩]

\setguidecanvas{⟨*style controller name* ⟩**}**[⟨*comma separated canvas options* ⟩][⟨*comma separated additional options* ⟩]

\setguidecolor{⟨*sttyle controller name* ⟩**}**[⟨*comma separated color options* ⟩][⟨*comma separated additional options* ⟩]

\setguidefont{⟨*style controller name* ⟩**}**[⟨*comma separated font options* ⟩]

Template name: A template name can be name of a component or an element. to specify an element within a particular container/component, 'in' keyword can be used. For example, *author in headline* can be styled separately from *author in footline*. there should be a single

space around 'in'. Following is the list of template names that can be specified:
Outer elements

```
 headline
, footline
, leftsidebar
, rightsidebar
, miniframes
, logo

, title
, title in headline
, title in footline
, title in sidebar

, subtitle
, subtitle in headline
, subtitle in footline
, subtitle in sidebar

, author
, author in headline
, author in footline
, author in sidebar

, date
, date in headline
, date in footline
, date in sidebar

, institute
, institute in headline
, institute in footline
, institute in sidebar

, slide number
, slide number in headline
, slide number in footline
, slide number in sidebar

, title graphic
, section graphic
, subsection graphic
```

```
              , slide title
              , slide subtitle

              , section
              , section in headline
              , section in footline
              , section in sidebar

              , subsection
              , subsection in headline
              , subsection in footline
              , subsection in sidebar

              , subsubsection
              , subsubsection in headline
              , subsubsection in footline
              , subsubsection in sidebar

              , shaded section in headline
              , shaded section in footline
              , shaded section in sidebar

              , shaded subsection in headline
              , shaded subsection in footline
              , shadeds ubsection in sidebar

              , shaded subsubsection in headline
              , shaded subsubsection in footline
              , shaded subsubsection in sidebar }
```

While the names are meant to be self-explanatory, Some of the elements need some description. When no container is specified with *author,title,institute* etc. then the implicit container is taken as title slide. Similarly *section/subsectionn/subsubsection* with no container are taken as being associated with Section/Subsection marker slides.*section, subsection, subsubsection* in a container are the elements that will be used in navigational menu-bar to represent the section/sub-section/sub-sub-section of current slide while shaded version will be used to represent other section/sub-section/sub-sub-section as applicable.
TheInner elements:

```
\textit{              , block
                      , framebox
```

```
                    , imagebox
                    , alertedblock
                    , quiz

                    , blocktitle in block
                    , blocktitle in alertedblock

                    , normal
                    , highlight
                    , alert

                    , itemize item
                    , itemize subitem
                    , itemize subsubitem
                    , enumerate item
                    , enumerate subitem
                    , enumerate subsubitem
                    , description item
                    , description subitem
                    , descriptions ubsubitem

                    , caption in imagebox
                    , captionnameandnumber in imagebox

                    , toc
                    , section in toc
                    , subsection in toc
                    , subsubsection in toc
                    , shaded section in toc
                    , shaded subsection in toc
                    , shaded subsubsection in toc }
```

normal, highlight and alert specify text elements. 'item', 'subitem' and 'subsubitem' are items in first,second and third level nested list respectively. They can be in *itemize,enumerate or description* list. *toc* is the element specifying table of contents.
The symbol like elements and subselements are given as follows:

```
\textit{        navigational symbols
              , dimmed navigational symbols
              , modechange symbols
              , dimmed modechange symbolsn

              , current in miniframes
```

```
            , current subsection in miniframes
            , other section in miniframes
            , other subsection in miniframes

            , section projection
            , shaded section projection
            , section projection in toc
            , shaded section projection in toc

            , subsection projection
            , shaded subsection projection
            , subsection projection in toc
            , shaded subsection projection in toc

            , subsubsection projection
            , shaded subsubsection projection
            , subsubsection projection in toc
            , shaded subsubsection projection in toc

            , itemize item projection
            , itemize subitem projection
            , itemize subsubitem projection

            , enumerate itemp rojection
            , enumerate subitem projection
            , enumerate subsubitem projection

            , description item projection
            , description subitem projection
            , description subsubitem projection}
```

dimmed templates of symbols specify style for symbols when a symbol is not hovered upon. *projection* templates are to describe the marker of corresponding element. For example, *shadedsubsectionprojection* is a template for marker preceeding a subsection when that subsection is shaded. It is in context of *toc*. While *itemize item projection* is to specify the marker of an itemized item.

Style controller name: These are the names of the style-controller that affect the large parts of the presentation as shown in figure **??**. Theses can be: *Outer , Inner , Outer Elements , Inner Elements , Title Like , Normal Text , Alert Text , Highlight Text , Item , Subitem, Subsubitem , Item Projection , Subitem Projection , Subsubitem Projection* .These are the only keywords with first letter capitalized.

Canvas Options: Options are specified as a comma separated list of options. An option consists of ⟨*option name* ⟩=⟨*value* ⟩. For example **\setcanvas}**[*border=thin,padding=none*]
Following are the canvas options:

1. *border* : It can take values - *thin/tick/medium/none* or it can be a number specifying the width of the border
2. *padding* : Padding thickness can be specified as *thin/tick/medium* or a number. Left, top, right, bottom paddings can be of different thickness.They can be specified in the same order with '!' as separator. For example, 'thin!thick!thick!2'.Not all of the four paddings are needed to be specified,i.e. a specification between two '!' can be empty. As an example, '!thin!!' will specify only top padding.

An additional option can also be specified with canvas which is - *shape*, that can take values *circle, tick, square, rectangle, triangle.* It is valid with symbol elements like projections for list items or section etc.

Color Options: A color in *SLATE* is the name of the color or code of the color in *rgb, rgba, hsl, hsla* encodings or 'transparent'. Valid color names are: *red , orange, yellow, green, blue, purple , brown, lightred, lightorange, lightyellow, lightgreen , lightblue , lightbrown, darkred, darkorange, darkyellow, darkgreen, darkblue, darkpurple, darkbrown , white lightgrey , grey, darkgrey, lightcharcoal charcoal, darkcharcoal, black*

The code can be specified as 'rgb:x:y:z' or 'rgba:x:y:z:a' or 'hsl:x:y:z' or 'hsla:x:y:z:a' where x,y,z,a are numbers between 0 to 255. Following are the color options:

1. *border* : Value can be any color
2. *padding* : Value can be any color or colors seperated by '!' to denote color of left, top, right, bottom padding respectively.

Two additional options can also be specified which are valid with symbol-like elements - *fill*, that takes a color and *alpha* that takes a floating number between 0 and 1 to specify opaqueness of the symbol, 0 being completely clear.

Font Options:

Following are the font options:

1. *size* : It can take values - tiny/scriptsize/footnotesize/verysmall /small/normalsize/large/verylarge/huge
2. *shape*: Value can be normal/bold/italic/bolditalic.

3. *align*: left/right/center/justifycan be given as value.

4. *family*: It takes the name of a font-family.

B.1.3 Layout

Apart from choosing a layout theme, layout of a component can be customised by the following commands :

\setlayout{⟨*component name* ⟩**}{**⟨*layout-content* ⟩**}**

The components whose layout can be specified are: *headline, footline , right sidebar , left sidebar, navigational symbols, modechange symbols , title slide , toc slide , atbeginsection slide , atbeginsubsection slide atbeginsection slide* specifies how a section marker slide must look like. Similary *atbeginsubsection slide* is for marking beginning of subsections.

Layout-content: Layout content can be specified by inserting various elements using the following command: **\insert{**⟨*element name* ⟩**}**[⟨*comma separated options* ⟩]
Various elements that can be inserted are: *navigation vertical,navigation , section navigation vertical ,section navigation , subsection navigation vertical , subsection navigation , tree section , tree subsection , section , subsection , subsubsection, logo , slide number and total , slide number , title , subtitle, author , institute , date , image , slide navigation back forward , support , zoom , quiz, scribble, toc* The valid inserts in a component are in accordance with the figure 3.1.

Layout Options Following are the layout options to be used only when *toc* is inserted:

1. *pausesections* : It can true or false. It instructs whether to reveal table of contents section-wise or as a whole.

2. *pausesubsections* : It can true or false. It can make table of contents to reveal subsection-wise, stopping after every subsection and displaying next subsection on the next 'tick.

3. sectionstyle: A section in a toc can be instructed to be hidden or shaded or normally visible by giving value *hide/shade/visible* respectively. This option specifies how a section in current section and other sections should be displayed by specifying style for each separated by '!'. For example, 'show/hide' will make the current section to be visible in the toc and other sections to be invisible.
   itemsubsectionstyle: It specifies what the current subsection, other subsections of the current section and other sections should be displayed. It is specified iby '!' separated styles in the above specified order. For example, 'show/shade/hide' will display the currebt subsection in toc as normal, other subsections of current section will be shaded and other sections will be completely hidden.

The elements will be inserted from top to bottom in the order in which they have been specified. To change the arrangement, following command can be used.

\begin{columns**}**
\begin{column**}**[⟨*comma separated column options* ⟩]**{**⟨*width* ⟩**}**
. . . ⟨*various insert commands* ⟩
. . .
\end{column**}**
. . .
\begin{column**}**[⟨*comma separated column options* ⟩]**{**⟨*width* ⟩**}**
. . . ⟨*various insert commands* ⟩
. . .
\end{column**}\end{**columns**}**

A column specify the elements to be stacked vertically while columns are arrange side by side. It is mandatory to specify the width of the column which is of the format 'x%' where x specify the width of column relative to the component.

Column options can be:

- *align*: Alignment of the columns with respect to each other. Value can be *t/b/c* corresponding to top, bottom and center alignment.
- *placement*: Placement of the elements in that column. Values can be *left/right/center*

B.2 Information

This part gives the meta information about the presentation. It has the following commands to specify titile, subtitle, authors, institute and date of the presentation. It also contains the commands to give the images that must be used in marker slides (Title/Section/Subsection). been specified. To change the arrangement, following command can be used.

\title{⟨*title of presentation* ⟩**}**
\subtitle{⟨*subtitle of presentation* ⟩**}**
\authors{⟨*comma separated author names* ⟩**}**
\institute{⟨*comma separated instittes in the order of author names* ⟩**}**
\date{⟨*date of presentation* ⟩**}**
\titlegraphic{⟨*absolute file name* ⟩**}**
\sectiongraphic{⟨*absolute file name* ⟩**}**
\subsectiongraphic{⟨*absolute file name* ⟩**}**

Currently these commands are not taking in-line style commands to bold/italicize etc.

B.3 Content

this is the section where actual content is specified. It contains the commands that specify various slides. The slide sequence can be logically divided by following commands to start a section/sub-section/sub-sub-section.

\section{⟨*short name* ⟩**}{**⟨*long name* ⟩**}**
\subsection{⟨*short name* ⟩**}{**⟨*long name* ⟩**}**
\subsubsection{⟨*short name* ⟩**}{**⟨*long name* ⟩**}**

Short name is mandatory. It will be used everywhere in navigation menu-bars or table of contents. Long name is the name used on Section marker slide. If it's not given, then the short name will be used instead.

B.3.1 Slide

A slide can be specified using the following command:

\begin{slide**}**[⟨*slide options* ⟩]**{**⟨*slide title* ⟩**}{**⟨*slide subtitle* ⟩**}**
...
⟨*slide-content* ⟩
...
\end{slide**}**

textbfSlide Options:

Following options are available with slide command

- *type*: It specifies the slide type. It can be *plain* when all Outer components are suppressed and content is give the full screen to be displayed. If it is not specified, type is *normal*.
- *supportbin*: It takes the value as some label name of a support-bin. That support-bin will then be associated with this slide which can be displayed in Supportbin mode.
- *quiz*: It takes a label name of a quiz. That quiz will be associated with this slide that will be displayed in Quiz mode while presenting.

textbfSlide Content:

Following are the types of content element that can be thee in a slide.

- Block: This element contains text with a title.
- Alertedblock: Same as the block, but its title will be in alerted text style

- Framebox: It is to frame the text.
- Imagebox: This element contains an image/graphic. It also specifies options for the width, height or scaling of the image.
- Enumerate/Itemize/Description: These are to contain text in a list format. Enumerate mark the items by numbers, itemize draws bullets as markers and description is a way to give definition lists. These lists can be nested in one another up to three levels.
- Smartdisplay: These are to contain text of a list in various graphical formats.

Following commands are used to describe these elements:

```
\begin{block}<⟨element-animation ⟩>
\title{⟨title of block ⟩}
⟨text-content ⟩
\end{block}

\begin{alertblock}<⟨element-animation ⟩>
\title{⟨title of block ⟩}
⟨text-content ⟩
\end{alertblock}

\begin{framebox}<⟨element-animation ⟩>
⟨text-content ⟩
\end{framebox}

\begin{imagebox}<⟨element-animation ⟩>[⟨image options ⟩]
\includegraphics{⟨absolute file name ⟩}
\caption{⟨caption of image ⟩}
\end{imagebox}

\begin{enumerate}
\item<⟨element-animation ⟩>⟨text-content or another list ⟩
...
\end{enumerate}

\begin{itemize}
\item<⟨element-animation ⟩>⟨text-content or another list ⟩
...
\end{itemize}

\begin{description}
\item<⟨element-animation ⟩>[⟨definition word ⟩]⟨text-content or another list ⟩
...
\end{description}

\begin{smartdisplay}<⟨smart-display type ⟩>
⟨list-content ⟩
\end{smartdisplay}
```

Smartdisplay type can be any of the *circle-full, circle-half, circle-cone, plus-equal-horizontal, plus-equal-vertical, plus-minus, paragraphlist, checkbox-comparision, containerlist, framedlist.*

Animation specification is a comma separated list of number or range of numbers specified using hyphen. For example, '¡1,4,6-8,9¿'. It specifies the 'ticks' at which element will be visible. At all other 'ticks' element will not at all be there on the slide. To change this effect and make an element stay at other ticks as well, but hidden the keyword *covered* can be used after specifying 'ticks' with a preceding '—'. For example '¡1,5,7—covered¿' will make an element visible at 'ticks' 1, 5 and 7. On all other 'ticks element will ne transparent. It will occupy space but won't be visible.

Other commands to specify for text animation are as follows

\visible<⟨*animation ticks* ⟩>**{**⟨*text-content* ⟩**}**
\invisible<⟨*animation ticks* ⟩>**{**⟨*text-content* ⟩**}**
\only<⟨*animation ticks* ⟩>**{**⟨*text-content* ⟩**}**
\alt<⟨*animation ticks* ⟩>**{**⟨*text-content for ticks* ⟩**}{**⟨*text-content for other ticks* ⟩**}**
\alt<⟨*animation ticks* ⟩>**{**⟨*text-content before ticks* ⟩**}{**⟨*text-content at ticks* ⟩**}{**⟨*text-content after ticks* ⟩**}**

visible command specifies 'tick' at which text will be visible. At all other 'ticks' it will be covered. invisible does the opposite of visible. only command is like visible but text will not be there on other 'ticks' and hence will not take any space. alt command specifies two texts, one that will be present at specified 'ticks' and other will be present at other times. temporal command can be used to specify different text that must appear at the specified ticks, before the biggest of specified ticks and after the given 'ticks'.

Text content is any character that is not a special character (<, >, {, }, [,], \). Any special character that is to be part of the text must br preceded by a backslash. A backslash followed by another backslash is to enter a new line in the text. All white space characters in the text are mapped to a single space. Any part of the text can be surrounded by in-line style commands which are to make the corresponding text bold or italic or underline or to alert or to highlight or to change font-color. The commands are **\alert**,**\highlight**,\textit,\textbf,**\underline**,**\color**⟨*color name* ⟩ These in-line style commands can further have animation specification. For example, '**\alert**<4,5>I am alert at 4 and 5 will make this text alert at 'ticks' 4 and 5. At all other 'ticks' when it is visible, it will be displayed as the normal text.

Arrangement of the elements in a slide can be specified by using *Columns* command as explained earlier in Layout subsection.

B.3.2 Supportbin

A supportbin can be specified by command:

\begin{supportbin**}{**⟨*label* ⟩**}**
...
⟨*content same as in a slide* ⟩
...
\end{supportbin**}**

Content is same as that in a slide. But the arrangement and animation of the content are ignored. Elements in a supportbin will be displayed as a grid in Supportbin mode that can be clicked to zoom.

B.3.3 Quiz

A quiz can be specified by the following command:

\begin{quiz**}{**⟨*label* ⟩**}**
\question{⟨*question text* ⟩**}**[⟨*question type* ⟩]
\corectanswers{⟨*comma separated possible correct answers* ⟩**}**
...
\answeroption{⟨*one of the option for mcq* ⟩**}**
...
\expalanation{⟨*explanation of answers* ⟩**}** ...
\end{quiz**}**

Question type can be *simple* or *mcq*. If none is specified, *simple* is taen. *correctanswers* specify the answers against which an answer submitted during quiz will be checked. If question is of type *mcq*, then all the spacified options using *answeroption* command will be displayed in a dropdown. The explanation givenusing *explanation* command will beshown when the button demanding the explanation will be clicked during the quiz.

Appendix C

A sample *SLATE* file

This is one of the sample files that were used to test *SLATE*.

```
\begin{presentation}

\begin{appearance}
    \usestyletheme{sober}
    \uselayouttheme{infolines}
    \setguidecanvas{Outer Elements}[padding=4!4!4!4]
    \setlayout{toc slide}{
        \insert{toc}[sectionstyle=show!show, subsectionstyle=show!show!show]
    }
\end{appearance}

\begin{information}
    \title{SLATE: Slide-Making in Web-Age}
    \authors{Ritu Kundu }
    \date{08th of March, 2015}
\end{information}

\begin{content}

\section{Introduction}
\subsection{Motivation}

\begin{slide}
    \begin{framebox}<1>
        To 'modernize' the slide-making and presentation delivery process \\
        in academic sphere.
    \end{framebox}
    \begin{smartlist}{plus-minus}<2>
        \item Latex
            \subitem Most Commonly Used
        \item
            \subitem Not for Dynamic or Interactive Content
    \end{smartlist}
    \begin{smartlist}{plus-minus}<2>
        \item Web-Applications
            \subitem Suitable for Dynamic and Interactive Content
        \item
            \subitem Requires HTML/CSS/JavaScript coded documents
    \end{smartlist}
    \begin{smartlist}{plus-equal-vertical}<3>
        \item Document in \\ Latex like language
        \item Display \\in \\web-browser
        \item SLATE
    \end{smartlist}
\end{slide}
```

```
\subsection{Current Presentation System}

\begin{slide}
    \begin{smartlist}{containerlist}
        \item Slideware \\Tools
            \subitem WYSIWYG editors
            \subitem Support for multimedia \\and animation
            \subitem !Temporal linearity of slides \\while delivering
            \subitem !Close-coupling of content and \\layout
            \subitem !Inability to support interactivity
        \item LATEX, \\Beamer and \\PGF/TikZ
            \subitem Unmatched typesetting quality
            \subitem Separation of content and layout
            \subitem Better portability because of the \\PDF format
            \subitem Convenience in expressing \\mathematical content
            \subitem !Support for animation not \\as good
            \subitem !Absence of interactivity
        \item Web-based\\ Presentation \\tools
            \subitem Robust support for multimedia \\including audio-video
            \subitem Support for conveniently creating richly \\interactive environments and dynami
content.
            \subitem Portability and easy \\accessibility
            \subitem !Require rigorous coding from the \\users.
    \end{smartlist}
\end{slide}

\subsection{Objectives}

\begin{slide}
\begin{columns}
    \begin{column}{0.6%}
    \begin{smartlist}{framedlist}
        \item Primary
            \subitem Input: Latex-like
                \subsubitem Reuse
                \subsubitem No need to learn new languages
            \subitem Output: Browser-displayable
                \subitem Dynamic/interactive content, portable and accessible
        \item Secondary
            \subitem Separation of 'Appearance' from the 'Content'
            \subitem Provision for Rich Animations
            \subitem Provision for Interactive Content like Quizzes
            \subitem Provision for breaking linear/sequential flow
            \subitem Provision for Additional "Back-up" Information
            \subitem Provision for "On-Demand" Zooming
```

```
            \subitem Provision for "On-Demand" Zooming
            \subitem Provision for "Scribbling"
            \subitem Provision for "Smart Lists"
    \end{smartlist}
    \end{column}
\end{columns}
\end{slide}

\section{Design & Implementation}
\subsection{Elm}

\begin{slide}
\begin{columns}
    \begin{column}{0.5%}
    \begin{smartlist}{paragraphlist}
        \item What is Elm?
            \subitem A new language for web-development which
                    \\appeared in 2011
            \subitem A functional reactive language for interactive
                     \\applications
            \subitem Compiles down to a combination of HTML, CSS and JS code
            \subitem import Mouse
                     \\main = lift asText Mouse.position
                     \\
                     \\A complete Elm program that prints and
                     \\updates the co-ordinates of the mouse continually.
    \end{smartlist}
    \end{column}

\end{columns}
\end{slide}
\begin{slide}
\begin{columns}
    \begin{column}{0.5%}
    \begin{smartlist}{paragraphlist}
        \item Why Elm?
            \subitem Much fewer Lines of Code
            \subitem Functional nature and Strong Static Type system
            \subitem Convenient and efficient build-in data-structures
                    \\in Elm (like dictionaries, lists and records)
            \subitem Excellent support for interactivity
    \end{smartlist}
    \end{column}
\end{columns}
\end{slide}
```

```
\section{Conclusions}
\begin{slide}
\begin{columns}
    \begin{column}{0.7%}
    \begin{smartlist}{framedlist}
        \item SLATE is an attempt to modernize Beamer for building presentations
\subitem Input is inspired from Beamer but slides are displayed in a web-browser.
        \item It provides more power to facilitate interactivity.

            \subitem Quizes
            \subitem On-demand zooming
            \subitem Scribbling
        \item It can be extended in many ways to make it a really powerful
            and \\convenient tool that could be preferred over
            contemporary \\presentation software.

    \end{smartlist}
    \end{column}
\end{columns}
\end{slide}

\begin{supportbin}{themeDiagrams}
    \begin{imagebox}[width=1282, height=551]
        \includegraphics{Default.png}
    \end{imagebox}
    \begin{imagebox}[width=1284, height=556]
        \includegraphics{sober.png}
    \end{imagebox}
    \begin{imagebox}[width=1290, height=569]
        \includegraphics{Striking.png}
    \end{imagebox}

\end{supportbin}

\begin{quiz}{quiz1}
    \question{What is the name of this tool?}
    \correctanswers{Slate}
    \explanation{It is on the Title Slide. :)}
\end{quiz}
\end{content}

\end{presentation}
```

Bibliography

[1] Leslie Lamport. LaTeX: A document preparation system. 1986.

[2] L. Lin, R. K. Atkinson, and et al. Using animations and visual cueing to support learning of scientific concepts and processes. *Computers Education*, 56(3):650–658, April 2011.

[3] Mike Unwalla FISTC. Latex: an introduction. *Communicator Springer*, page 33, 2006. URL http://www.techscribe.co.uk/ta/latex-introduction.pdf.

[4] Parsing expression grammar. URL http://en.wikipedia.org/wiki/Parsing_expression_grammar.

[5] The not so short introduction to latex 2.