# Integrated intelligent LEARNing environment for Reading and Writing

## D4.2 – Content Adaptation and Presentation Module

| | |
|---|---|
| Document identifier | **2014_09_29_ILearnRW_D4.2_Content_adaptation_and_presentation_module_updated** |
| Date | **2014-09-29** |
| WP | **WP4** |
| Partners | **NTUA, DOLPHIN, UOB, LBUS** |
| WP Lead Partner | **NTUA** |
| Document status | **V0.9 / Final - Updated** |

| | |
|---|---|
| **Deliverable Number** | D4.2 |
| **Deliverable Title** | Content adaptation and presentation module |
| **Deliverable version number** | V0.9 |
| **Work package** | WP4 |
| **Task** | Task 4.2 Content Adaptation and Presentation Mechanisms |
| **Nature of the deliverable** | Report (R) |
| **Dissemination level** | Public (PU) |
| **Date of Version** | 2014-09-29 |

| | |
|---|---|
| **Author(s)** | Mattias Karlsson, Chris Litsas, Evmorfia Argyriou |
| **Contributor(s)** | |
| **Reviewer(s)** | Antonios Symvonis |
| **Abstract** | This deliverable describes the Content Presentation and Adaptation Module which is part of the iLearnRW system. Its main feature is to provide the basic API that contains methods to display the file to be read by the user. The module annotates the content by taking into account the user's profile and by following specific rules defined for each profile entry problem. |
| | |
| **Keywords** | iLearnRW, Content adaptation, Content presentation, API, Reader. |

**Document Status Sheet**

| Issue | Date | Comment | Author |
|---|---|---|---|
| v01 | 2014-02-17 | Basic Description of Annotation-Presentation Component | Mattias Karlsson |
| v02 | 2014-02-20 | Use Cases - API Requirements | Chris Litsas, Evmorfia Argyriou |
| v03 | 2014-03-10 | Text Adaptation Web Service | Chris Litsas |
| v04 | 2014-03-26 | API Modules Description | Evmorfia Argyriou |
| v05 | 2014-03-28 | Internal Review | Antonios Symvonis |
| v06-final | 2014-03-30 | Final version – Incorporating Internal Reviewer's Comments | Evmorfia Argyriou |
| v07-updated | 2014-09-15 | Updated version of the API | Evmorfia Argyriou |
| v08-updated | 2014-09-20 | Internal Review | Chris Litsas, Antonios Symvonis |
| v09- final | 2014-09-29 | Final version – Incorporating Internal Reviewer's Comments | Evmorfia Argyriou |

**Project information**

| | |
|---|---|
| Project acronym: | **ILearnRW** |
| Project full title: | **Integrated Intelligent Learning Environment for Reading and Writing** |
| Proposal/Contract no.: | **318803** |

**Project Officer: Krister Olson**

| | |
|---|---|
| Address: | L-2920 Luxembourg, Luxembourg |
| Phone: | +352430134332 |
| E-mail: | krister.olson@ec.europa.eu |

**Project Co-ordinator: Noel Duffy**

| | |
|---|---|
| Address: | Dolphin Computer Access Ltd. Technology House, Blackpole Estate West, Worcester, UK. WR3 8TJ |
| Phone: | +01905 754 577 |
| Fax: | +01905 754 559 |
| E-mail: | noel.duffy@yourdolphin.com |

# Table of Contents

# 1. Introduction

In this report, we describe the developed Content Adaptation and Presentation (or CAP for short) packages of the iLearnRW system which aim to present the content of a text file by appropriately adapting it to the user's needs. In other words, these mechanisms take into consideration the profile of a user and try to present the text in an appropriate way such that potential problematic words (or parts of them) are distinguished and emphasized by several diverse ways in the text (e.g. modifying the text colour of the problematic part of a word, highlight the problematic part, etc.).

The content adaptation and presentation packages are implemented in the form of application programming interfaces (APIs) so that they can be employed/ integrated into several different devices and applications (standalone applications, applets, mobile applications, e-book readers, serious games, etc.). At this point, we have to emphasize that our aim was not to create a reader application that loads a text file and displays it appropriately on a screen. Instead, we present a content adaptation and presentation API that allows the generation of multiple different reader applications some of which are described in the form of use-cases in Section 2.

The content adaptation and presentation API consists of several modules that run either on the server's side or on client's side. The basic modules of the API are the following:

- The **Text Adaption Web Service** that runs on the server's side and takes as input a text file from which it generates an HTML file and a JSON file that contains information about each word of the text (e.g., id, difficulty, etc.).
- The **Presentation Rules Module** that specifies the presentation style of each particular entry of the profile of a user.
- The **Text Annotation Module** that parses the HTML and JSON files generated by the server and creates an annotated HTML file that includes the presentation style of each word of the text.
- The **Smart Display Module** that displays the annotated HTML file.
- The **Text-To-Speech** engine that provides additional help to the user during a reading session.

This report is structured as follows: In Section 2, we introduce some possible use-cases of a reader application. In Section 3, we present the CAP API requirements based on the use-cases presented in Section 2. Section 4 describes the text adaption web service and Section 5 describes how the client's side modules of the iLearnRW system interact in order to present a text in a friendly way in the screen of a user. Section 6 presents an example demonstrating the usage of the CAP API in a web application. Section 7 provides an updated version of the CAP API. The updated version includes some minor modifications that were necessary in order to address some difficulties that were revealed during the implementation of a reader client.

# 2. Use-Cases (Reader Scenarios)

In this section, we introduce some possible use-cases of a reader application. These include both the basic features of a reader but, also more sophisticated ones that will help to specify the requirements in order to build the CAP API.

## 2.1. A Simple Reader

The scope of a simple reader is to display text on the screen. This implies that it does not take into consideration any possible dyslectic difficulties or the profile of the child that uses the application. In brief, some basic operations of such a reader may include: (i) loading an input file, (ii) displaying the content of the input file and (iii) supplementary functionalities such as highlighting on words/sentences, popup menus displaying information about selected words, etc. (see Figure 1).


The input file of such a reader may be either a text file or an HTML file containing tags that specify the presentation style of letters, words or sentences of the text. Basic functionalities that the reader should support regarding this step of the process include the following:

- Navigation in the file system and search for documents.
- Selection of a document from the library.
- File filters so that only text and HTML files are selected and loaded to the application.

Such a reader should also provide basic functionalities that allow the user to (i) modify the presentation settings such as font-size, font-family, margins, etc. (ii) navigate through the text pages and (iii) perform searches within text. More precisely, the reader should support the following operations:

- Modification of page style (font-size, font-family, margins, etc.).
- Modification of screen orientation.
- Navigation through pages (quick jump to a page).
- Search for a word or a sentence in the text.

A simple reader may also support the presentation of the text with highlighting on words/sentences. For this reason, it should be equipped with buttons that start, pause or stop the highlighting operation over the text and sliders that control the highlighting speed. It should also enable the selection of words of the text and should provide information about these particular words. For instance, if a word is selected, the reader may open a popup window that provides:

- The syllabification of the word.
- Synonyms of the word.
- A button that activates the text to speech engine in order to hear the correct pronunciation of the word.
- A button that adds the word to the list of “tricky words” (The list of “tricky words” is a user specific list of words that present increased difficulties to a particular user).

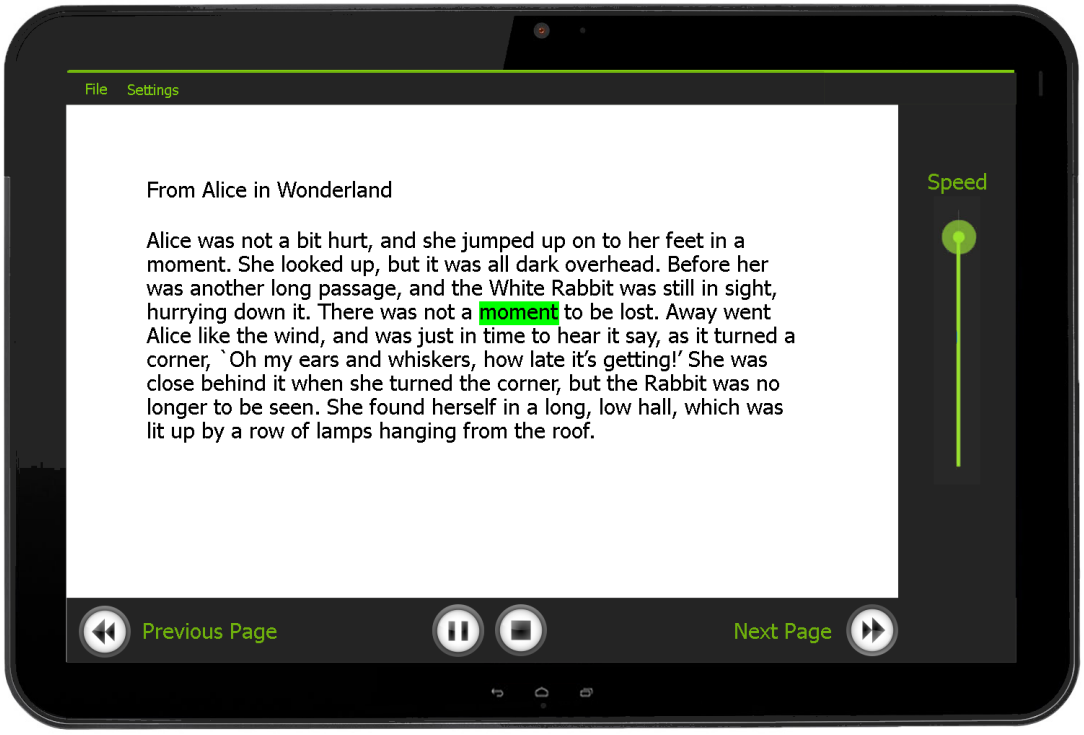


**Figure 1. A simple reader application that displays a text file with highlighting over the words.**

## 2.2. Special User-Aware Reader

A user-aware reader displays the content of a text file in such a manner that only specific problems (out of a list of predefined problems) are selected and highlighted during the presentation of the text. In brief, such a reader should support the following operations that will be described in detail later in this section:

- Loading of the input text file that has to be presented.
- Annotation of the input text according to the user's specifications and creation of an annotated HTML file.
- Displaying the annotated HTML file.
- Supplementary operations on the text (e.g. select words, etc.)


Obviously, the first step is trivial. The reader has to take as input the text that has to be presented. However, the next step, i.e., the text annotation, is the most complicated part of the above procedure. The reader has to allow the selection of particular problems and allow the user to determine their presentation style. For example, the reader should allow the user to select:

- A set of problems.

- A linguistic category (i.e. all problems inside the category).
- Only problems that are inside the Working Indices of the user profile.
- A set of problems on which the user has a severity value greater than a threshold value (e.g. highlight only problems from the child's profile with severity value greater than two).
- Problem (or set of problems) that will be highlighted even if the user does not have an issue with it (e.g. highlight problems even if in the child's profile their severity value is zero).

One way that the reader may choose to implement the above is to employ a screen similar to the one of Figure 2, which resembles to a table, in which each cell corresponds to a particular problem. The background color (i.e., green or yellow) of each cell represents whether the problem will be taken under consideration during the annotation or not. Pressing on a cell activates or deactivates the particular problem.

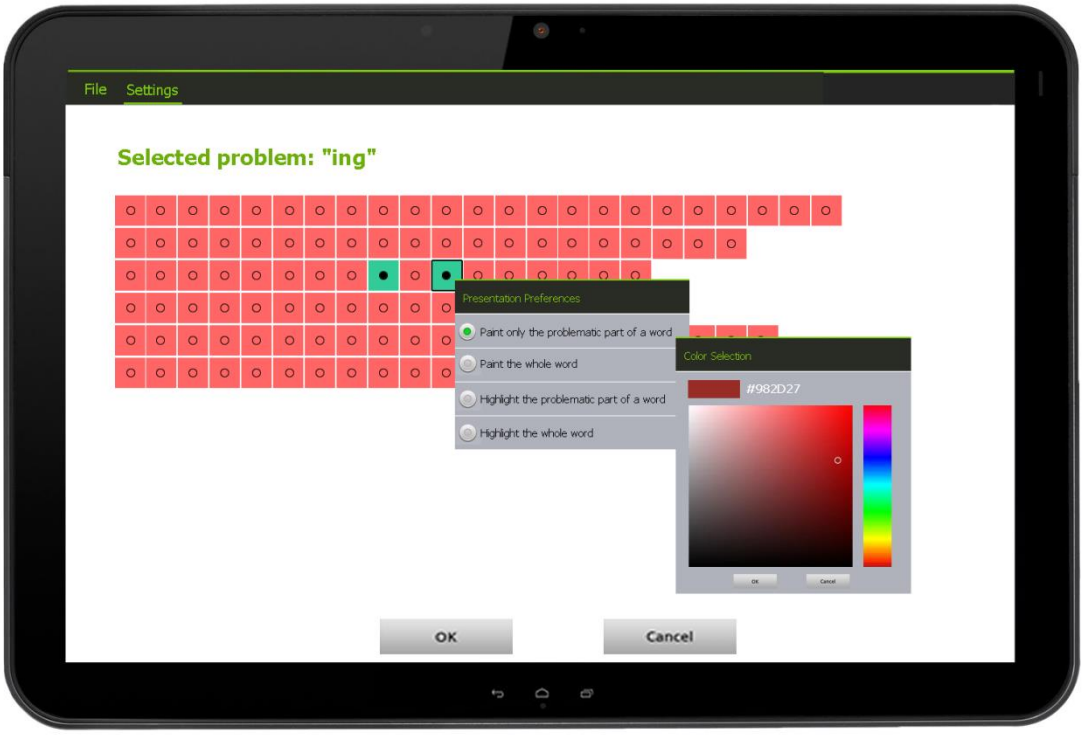


**Figure 2. Determining the problems on which the user focuses and the presentation rules for each particular problem.**

After the selection of the problems on which the presentation will focus, the user has to decide whether she modifies also the presentation rules for each problem. Otherwise, the presentation of the text will be performed based on the presentation rules currently stored in the system. Possible presentation rules may include the following:

- Paint only the problematic part of a word.
- Paint the whole word.
- Highlight the problematic part of a word.
- Highlight the whole word.

Long press on a cell may allow the user to modify the presentation rules of the particular problem (see Figure 2). This allows the user to modify either the presentation color or the presentation style (e.g.

paint the whole word instead of painting only its problematic part). For instance, the user is able to deploy the "*-ing*" overlay, which paints the problematic parts of all words relevant to the "*-ing*" difficulty with the background color "*red*".

According to the specifications of the user, the reader should annotate the text and create an HTML file that contains all the information about the presentation of each word or sentence. However, it is highly possible that in the case where a user selects a great number of problems on which she focuses simultaneously or if the text contains many difficult words that fall into the same problem or linguistic category, the resulting presentation will become inappropriate for a child with dyslexia (e.g. many highlighted or colored words in the text). For this reason, the presentation should be appropriately adjusted based on thresholds defined by the user or the system. For instance, if many words have to be highlighted in the same screen, one can decide to highlight only a percentage of them (e.g. 10%), which means that only the 10% of the most difficult words of the text that fall into the selected problems will be eventually presented. The reader should be able either to re-annotate the text before displaying the text based on these thresholds or it may display the text as produced by the annotation and let the user determine the thresholds that filter the presented document by moving appropriately sliders controlling these thresholds.

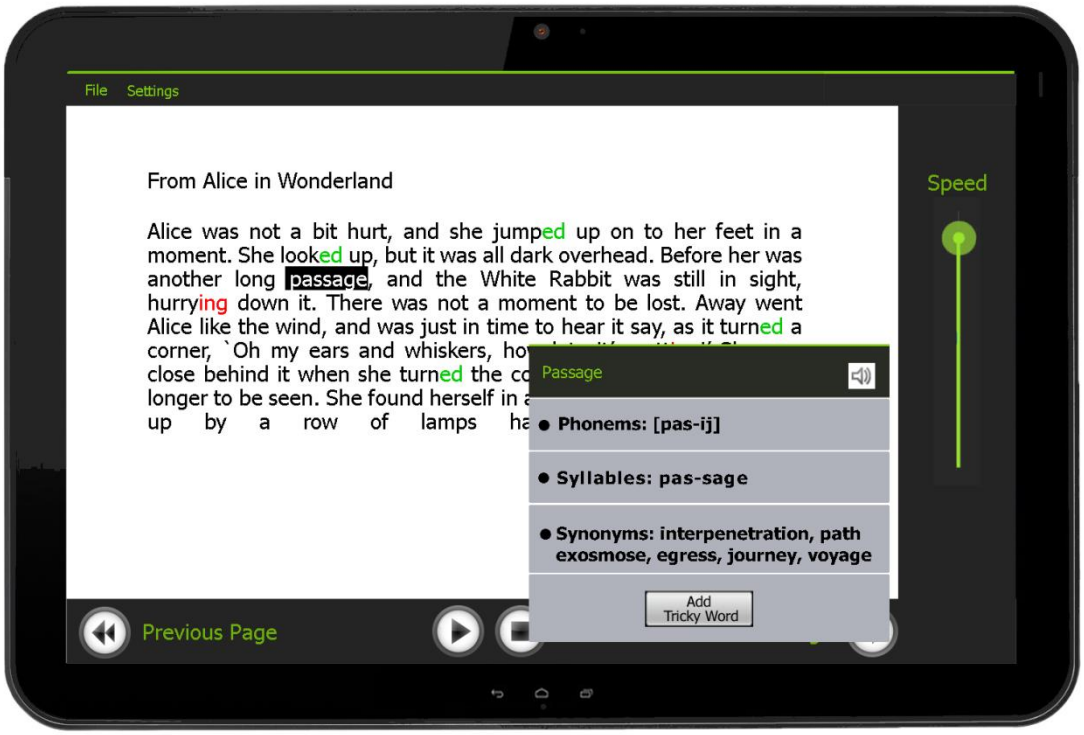


**Figure 3. Displaying a text using a special user-aware reader focusing only on "ing" and "ed" overlay. The word "passage" is selected by the user.**

Obviously, a user-aware reader should also support most of the functionalities of the simple reader described above such as modification of the presentation settings (e.g. font-size, font-family, margins, etc.), navigation through the pages, searching etc. It should also be equipped with buttons enabling the highlighting operation over the text. Finally, a user-aware reader should also support "on touch" events for word selection. A possible usage of such an operation is to open a popup window or present

to a separate panel of the application (see Figure 3) information about the word (as described in the case of the simple reader), activate the text to speech engine or add the word to the list of "tricky words" for the particular child.

A supplementary functionality of this reader may allow the user to store a record for the settings regarding the problems on which the user has focused and the presentation rules that he performed in order to use them in a next session with the same child. Optionally, the reader may allow the user to modify the severity values in the child's profile if needed.

## 2.3. Generic User-Aware Reader

A generic user-aware reader can be used to display text in several components and applications of the iLearnRW system (stand-alone applications, serious games, etc.). For instance, it can be used in a mini game in order to display the rules or the messages of the game. In this case, the reader may consist of a single window and simple buttons that allow the navigation through text pages. A generic user-aware reader may also support the highlighting of the words of the text. For this reason, it has to be equipped with buttons that start, pause and stop the highlighting and sliders that control the highlighting speed. It may also support text to speech operation (see Figure 4).

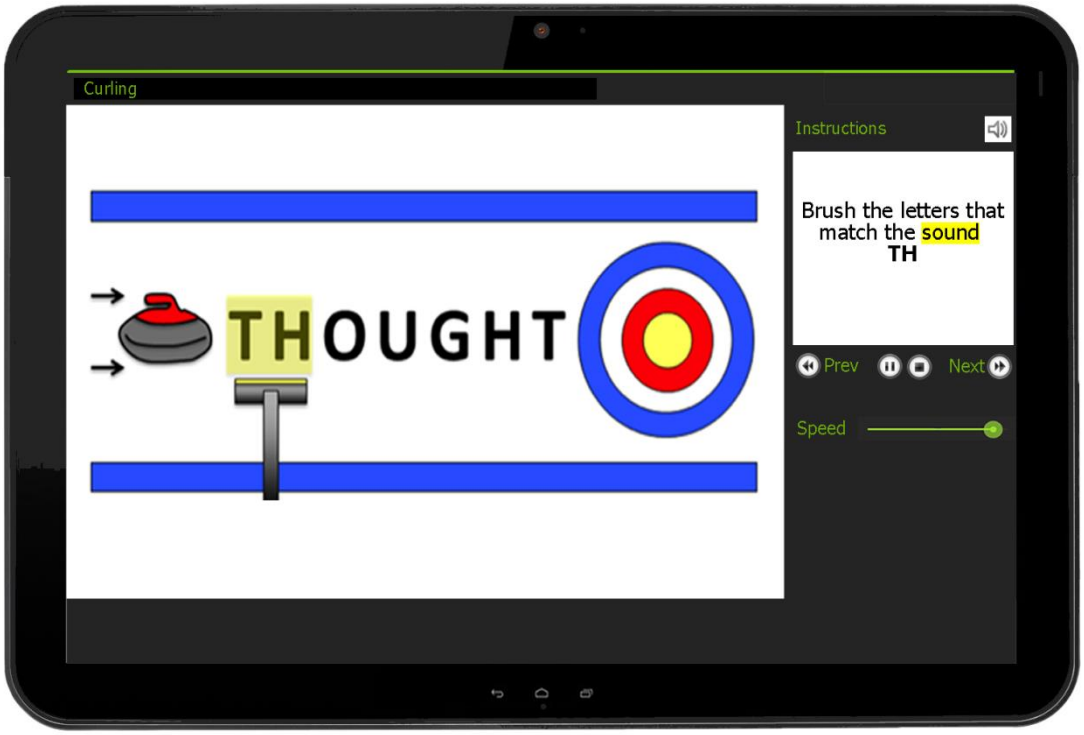


**Figure 4. A generic user-aware reader that displays the instructions of a mini-game with highlighting on words.**

# 3. Text Adaptation/Presentation Requirements

The goal of the Text Adaptation/Presentation API is to implement content adaptation and presentation mechanisms (which include text-to-speech) that conform to the user's profile and result in content presentation styles that are suitable for each individual user. The reader scenarios described in Sections 2.1-2.3 can be considered as the base in order to develop the CAP API of iLearnRW system. In the remainder of this section, we will describe in detail the requirements implied by the above use cases that the CAP API should meet.

## 3.1. User Profile Access/Modification

In order to adapt the presentation of a document to the user's needs, there should exist methods that allow loading the profile of a user.

## 3.2. Text analysis to find problematic parts of each word

Text analysis constitutes one of the most important tasks of the text adaptation procedure. During this step, the text will be processed and each word will be examined in order to identify possible problem(s) that a child with dyslexia may face and the exact position(s) that these problems are met in the word. This procedure is performed in the iLearnRW server. The API should provide methods that communicate with the server and make the text processing. Methods that calculate the difficulty of a word based on certain rules should be also included. More details about this procedure will be discussed in Section 4.

## 3.3. Selection of problems / Filtering

The user's profile, as it is supported by the iLearnRW system, consists of a large number of language related difficulties (or problems). For this reason, it is necessary to allow the tutor to select the problems on which she is going to focus. Based on the case scenarios of Sections 2.1-2.3, the API should provide methods that:

- Facilitate the selection of a set of problems.
- Allow the selection of a whole linguistic category (i.e. all problems inside the same category).

Based on the above methods, a reader application can implement the case where it is desired to focus only on problems that fall into the working indices of the user's profile or where it is desired to force the presentation of given problems even if their corresponding severity value is zero in the user's profile.

## 3.4. Presentation styles

The API should provide methods that determine the presentation style of each difficult word of the text. Consider for instance, the case where a child's profile indicates that she faces difficulty on the "*-ing*" suffix. Then, the user can deploy the "*-ing*" overlay. Possible presentation styles of word "playing" (that contains the "*-ing*" suffix) may include the following:

- Paint only the problematic part of a word. Then, the word "playing" will be displayed as "playing".

- Paint the whole word. Then, the word "playing" will be displayed as "playing".
- Highlight the problematic part of a word. Then, the word "playing" will be displayed as "playing".
- Highlight the whole word. Then, the word "playing" will be displayed as "playing".

## 3.5. Presentation Settings

Within any digital reader it is necessary to determine the presentation style. This task gains more importance in iLearnRW system, since our aim is to assist children with reading difficulties. A list of text adaption styles includes the following:

- **Font Family:** The API should provide methods that allow the user to modify the font face of the text. The system's default font is Tahoma.
- **Font Size:** The API should provide methods that allow the user to adjust the size of the text. Available values are 80%, 90%, 100%, 120%, 150% and 200%. Default value is 100%. These values correspond to "Very small", "small", "smaller", "normal", "larger", "large", "very large".
- **Line Spacing**: Line spacing corresponds to the amount of space between lines. The default value is set to 1.25.
- **Margins**: Margin is an offset value for the text width and should be implemented as a percentage value. Default is no margin (full screen size) and available values are 5%, 10% and 20%. Instead of using percentage values, more readable values can be used: no margin, cozy, comfy and roomy.
- **Background colour**
- **Foreground colour**
- **Display Panel Width**
- **Display Panel Height**

Also, it is required to have methods that allow the user to maintain history files with the user's preferences and history data (e.g. font-size, last-page viewed, etc.).

## 3.6. Highlighting

The CAP API should support the "highlighting'' operation. In the iLearnRW system, the highlighting operation can be considered by two different aspects. In the first case, our intention is to highlight at once, all words that contain key difficulties that match to the user's profile according to the presentation style settings (either by highlighting only the problematic part or the whole word). In the second case, highlighting will be considered as an assistant for reading. During the reading, a moving cursor (at normal reading speed) highlights the current word or parts of it. The user is able to start, pause or stop the highlighting operation and increase or decrease the highlighting speed. Thus, the API should provide methods that support all the above operations.

## 3.7. Text-To-Speech engine

The iLearnRW system employs text-to-speech in order to provide multimodal input to children and make it possible for children to access the text without having to read it. Thus, the API should provide methods that regulate the TTS engine such as:

- **Play/Pause:** Pause/resume playback. The default state of this setting is pause.
- **Stop:** Stop playback and close the listen mode.
- **Fast Forward/Jump forward:** Jump to previous sentence or word.
- **Rewind/Jump back:** Jump to previous sentence or word.
- **Speed slider:** Increase / decrease playback speed
- **Pitch slider:** Increase / decrease voice pitch
- **Voice dropdown:** Choose Text-to-Speech voice
- **Volume slider:** Increase / decrease volume

## 3.8. "On Touch event" methods

"On Touch event" methods aim to provide learning and assistive features to the user and are activated whenever a single word is selected using the standard android operation of tap and hold. The API should support (i) "on click methods" which are called whenever the user either touches an item, focuses upon the item with the navigation-keys or presses a button, (ii) "on long click" methods that are similar to "on click methods" but the action lasts for one second or more, (iii) "on touch" methods that are called when a touch event (press, release, etc.) is performed, and (iv) "on swipe" methods that are called whenever the user is wiping a finger across the screen of a tablet (e.g. swiping to navigate to the next page).

## 3.9. Navigation/search methods

The API should provide methods that allow the user to navigate within the text such as:

- Go to previous/next page
- Go to previous/next word
- Go to previous/next sentence
- Go to selected page

The API should also allow the user to search a word in a page.

## 3.10. Low priority (if time permits)

The following features are considered of low priority and will be supported only if time permits in a later release of the library.

- Chunking mode (see D3.5 Section A.1.1)
- Text information screen (see D3.5 Section A.1.2)
    - Dublin Core (title, author, etc…)
    - Text difficulty/readability
        - Score translated into readable word (e.g. "challenging")
        - # of tricky words

        - Interest age
        - Reading age
    - Text metric
        - # of words
        - # of sentences
        - # of paragraphs
        - # of headings
        - Average word length
        - Average sentence length
- Selection of Day-Night view.
- Screen orientation.
- Screen contrast (low, medium, high).
- Tricky words.
- Letter kerning / word spacing.
- Support of headings and relevant operations (e.g. go to previous/next heading).
- Table of contents window (with navigation by heading).
- Regulation of screen contrast (low, medium, high).
- Pictionary.

## 3.11. Outside scope of project

The following functionalities that may be considered useful by some readers fall outside the scope of the project.

- Book list / Library in Reader (see D3.5 Section A.2.1)
- Bookmarks / Notes / Highlight notes (see D3.5 Section A.2.3)
- General settings, for example: (see D3.5 Section A.2.4)
    - Brightness
    - Page turning methods (our text is “flowing content” and we don’t have a definition of a page)
- User help overlay (see D3.5 Section A.2.5)

## 3.12. Content Adaptation/Presentation API diagram

The content adaptation and presentation packages will be implemented in the form of web services and client modules so that they can be used from several different components and applications of the ILearnRW system (standalone applications, readers, serious games, etc.).

The content adaptation and presentation is structured based on the following tasks:

1. The user wants to read a text.
2. The Reader Client provides methods to select text from the client’s device (e.g. the SD card) or from the resource bank.
3. The Reader Client sends the text file and the profile of the user to the Text Annotation Module.
4. The Text Annotation Module sends the text to the Text Adaption Web Service.

5. The Server processes the text and generates two files (one HTML containing the text and a JSON file containing information about words).
6. The Text Annotation Module downloads the two generated files.
7. The Text Annotation Module asks the Presentation Rules Module if and how each word will be presented.
8. The Text Annotation Module generates an annotated HTML.
9. The Text Annotation Module feeds HTML to Smart Display Module.
10. The Smart Display Module is now ready for:
    a. User commands
        i. Start stop TTS
        ii. Navigation
        iii. Get / Set settings
    b. Text style modifications
    c. "Highlight" operation.

The above tasks are summarized in the following diagram:

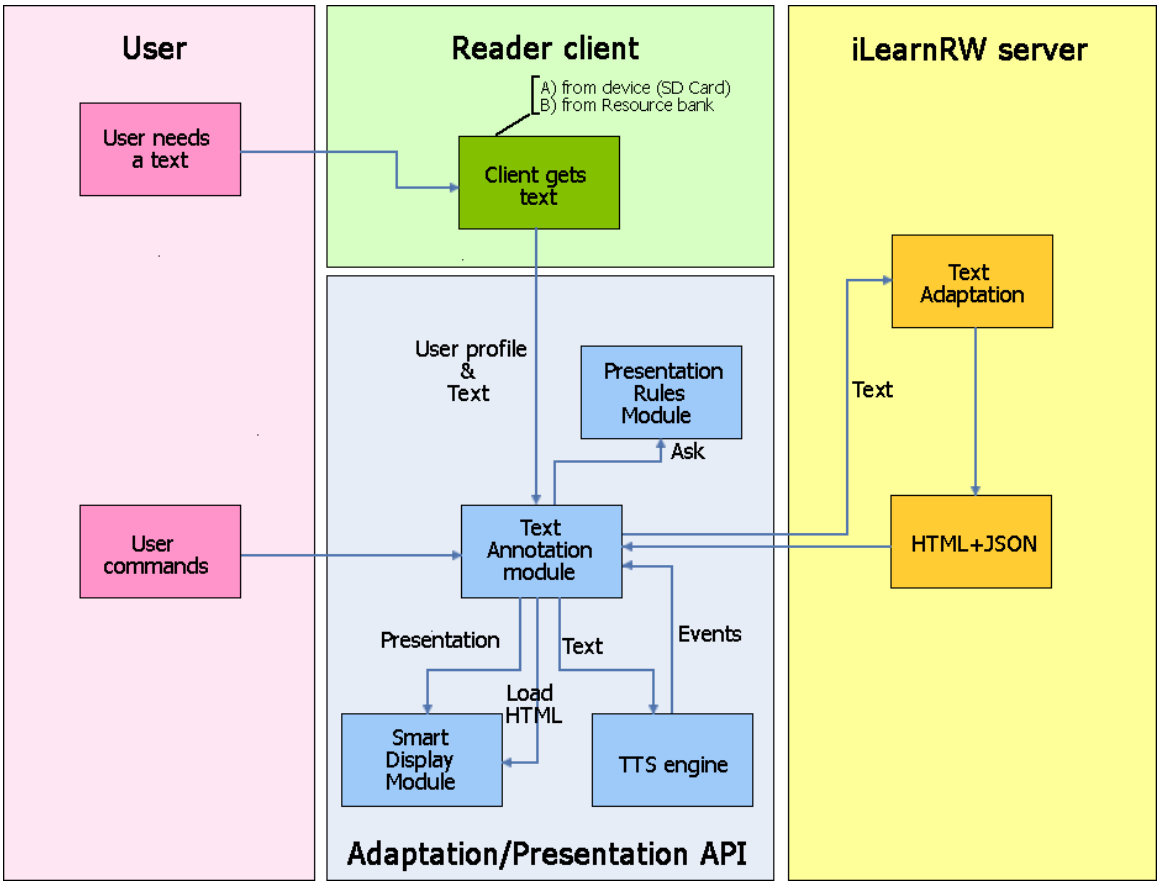


Figure 5. Overview Diagram for Content Presentation-Adaptation Component

In the remainder of this document, we will describe in detail the basic modules of the iLearnRW system.

# 4. Text adaption web service

The "text adaption web service" is a server-side component that processes the text from the user and prepares it for the presentation layer on the client side. This service marks-up the text on paragraph, sentence and word level and stores extra word information in a JSON file.

## 4.1. Workflow server process

Below is a simple workflow diagram describing how the server side processes the incoming text content resulting into two files.

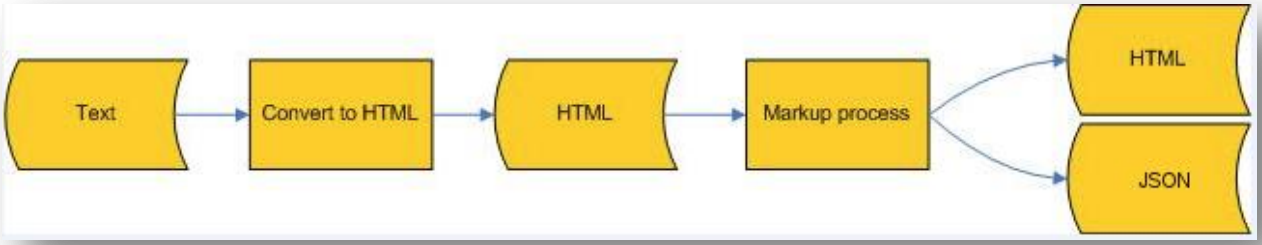


Figure 6. Workflow of the Process

## 4.2. Content conversion

The incoming text must be prepared and if needed converted to HTML. We have chosen HTML for the content as it contains the basic semantic structure we need for this project. HTML may contain information about headings, paragraphs and so on, which provider important information for the client and its user. Additional benefit of HTML is that it is easily displayed on different clients and platforms. There is no need for developing a special rendering component for these text – only methods to load and manipulate the loaded HTML in the web browser control.
We use the JSON format to store the data that are related to the entries of the HTML file. This format is well structured and the main benefit of this type of data is that it can be parsed immediately to a Java object.

The following are potential types of text converters:

- **TXT** – this converter wraps the plain text into HTML and it inserts <p> tags for all existing LF/CR. Headings will not be detected, as they do not exist in plain text. If we need headings to be detected, there are multiple ways to do this, but is at the moment out-of-scope.
- **EPUB** – this converter extracts the HTML documents in the EPUB file and merges these into one long HTML file. The resulting HTML file needs to be passed on to the HTML converter for further processing.
- **DOCX** – this converter extracts out the XML document and transforms it to HTML by using XSLT transforms and other methods. The resulting HTML file needs to be passed on to the HTML converter for further processing.
- **HTML** – this converter makes sure that the resulting HTML file can be processed further in the mark-up process. This module need to make sure that it is valid XHTML 1.0 encoded in

UTF-8 and it will also clean up segments that we do not support in this project. E.g. we may clean up multimedia objects, certain styling attributes and so on.

NOTE: Given the amount of time within this project, the development team will primarily focus on the TXT converter. If time permits, we may look at the other conversion modules.

## 4.3. Mark-up process

### 4.3.1. The HTML file

This process is where the server analyses the incoming text and runs the text classification. This module will basically do a sentence and word detection. Each sentence in the text will be marked up with a <sen> tag at the beginning of the sentence and close it with a </sen> tag. Each word in the text will be wrapped with <w> and </w> tags. Important is that each sentence and word contains an ID attribute for identification.
Important in this process is that we do not include "white space" characters and punctuation marks (e.g. . , ! ? : ; ) within the word mark-up.
One example:

```
<p id = p1>
<sen id = s3>
<w id = w30>Before</w> <w id = w31>her</w> <w id = w32>was</w> <w id = w33>another</w> <w id = w34>long</w> <w id = w35>passage</w>, <w id = w36>and</w> <w id = w37>the</w> <w id = w38>White</w> <w id = w39>Rabbit</w> <w id = w40>was</w> <w id = w41>still</w> <w id = w42>in</w> <w id = w43>sight</w>, <w id = w44>hurrying</w> <w id = w45>down</w> <w id = w46>it</w>.
</sen>
<sen id = s4>
<w id = w47>There</w> <w id = w48>was</w> <w id = w49>not</w> <w id = w50>a</w> <w id = w51>moment</w> <w id = w52>to</w> <w id = w53>be</w> <w id = w54>lost</w>.
</sen>
</p>
```

This example shows that space and period characters are not inside a word tag.

### 4.3.2. The JSON file

For each detected word in the text, more information from the text classification class is stored in a separate JSON file. This JSON file contains a list of all words in the text and this is where the text classification information is stored.
Note that, even if a word is repeated in the text, only one occurrence of the word exists in this JSON file. Note also, that these words are case sensitive, so the word "Boat" and "boat" would have different matching entries in the JSON file.

For example the word “There” (the fourth word in the above document) would result in the following entry in the JSON file:


```
"47":{
        "word":"there",
        "wordUnmodified":"There",
        "type":"Adverb",
        "syllables":[
          "there"
        ],
        "phonetics":"ðɛə",
        "stem":"there",
        "languageCode":"EN",
        "frequency":784528.0,
        "graphemesPhonemes":[
          {
            "grapheme":"th",
            "phoneme":"ð"
          },
          {
            "grapheme":"e",
            "phoneme":"ɛ"
          },
          {
            "grapheme":"re",
            "phoneme":"ə"
          }
        ],
        "userSeveritiesOnWordProblems":[
          {
            "category":4,
            "index":54,
            "matched":[
              {
                "matchedPart":"th",
                "start":0,
                "end":2,
                "matched":true
              }
            ],
            "userSeverity":0
          }
        ],
```

```
    "numberOfSyllables":1,
    "cvform":"-ccvcv-",
    "length":5
  },
```


Details:

- **word** : the word as it is on the iLearnRW dictionaries
- **wordUnmodified** : the word as it is inside the text (case sensitive)
- **type** : the word class (e.g. verb, noun, adverb, preposition, etc.)
- **syllables** : decomposition of the word into syllables
- **phonetics** : the phonetic transcription of the word
- **stem** : the stem of the word
- **languageCode** : the language of the word (English or Greek)
- **frequency** : describes how commonly a word appears in a typical text
- **graphemesPhonemes** : decomposition of a word into pairs of graphemes and their corresponding phonemes
- **userSeveritiesOnWordProblems** : this section contains a list of profile difficulties for the word
    - **category** : the number of the linguistic category of the problem matched
    - **index** : the index of a specific problem inside the above linguistic category
    - **matched** :a list with information of what is the part that matched (start, end)
    - **userSeverity** : the severity of the user to this problem, -1 if no user is specified
- **numberOfSyllables** : the number of syllables
- **length** : the word length in letters

## 4.4. Meta data in document

The resulting HTML document needs to store and retain certain meta-data from its original source document. These meta-data can be used in the reader client for different purposes. As the text adaption web service processes plain text, there is no requirement to provide these meta-data in the resulting HTML file. However, when possible, some meta-data must be carried over.
Basic set of Dublin Core:

- **dc:title**
  A name given to the resource, e.g. a document title
- **dc:creator**
  An entity primarily responsible for creating the resource. Can be either a person, author, organization or service.
- **dc:source**
  A related resource from which the described resource is obtained.
- **dc:subject**
  The topic of the resource.
- **dc:language**
  Text language of the resource. It is recommended to use RFC 4646 language tags.

- **dc:description**
  Description may include but is not limited to: an abstract, a table of contents, a graphical representation, or a free-text account of the resource.

The metadata are stored at the head section of the HTML as it is demonstrated in the following example:

```
…
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
        <title>From Alice in Wonderland</title>
        <meta name="dc:title" content="From Alice in Wonderland"/>
        <meta name="dc:creator" content="Lewis Carrol"/>
        <meta name="dc:source" content="Project Gutenberg"/>
        <meta name="dc:subject" content="novel"/>
        <meta name="dc:language" content="en-uk"/>
        <meta name="dc:description" content="Alice in Wonderland by Lewis Carroll is a fictional story about a young girl named Alice who spots an interesting white rabbit, muttering about being late, that she follows down a rabbit hole into a magical world."/>
    </head>
…
```

## 4.5. A full example

In the following we present a full example of the mark-up process. We show the text that is displayed in the browser, the contents of the HTML file and a segment of the corresponding JSON file:

**Text in browser:**

> **From Alice in Wonderland**
>
> *Alice was not a bit hurt, and she jumped up on to her feet in a moment. She looked up, but it was all dark overhead. Before her was another long passage, and the White Rabbit was still in sight, hurrying down it. There was not a moment to be lost. Away went Alice like the wind, and was just in time to hear it say, as it turned a corner, `Oh my ears and whiskers, how late it's getting!' She was close behind it when she turned the corner, but the Rabbit was no longer to be seen. She found herself in a long, low hall, which was lit up by a row of lamps hanging from the roof.*

The content adaption web service has processed the text content in the HTML file and marked up sentences and words. All sentence and word elements have unique ID attributes. These IDs are used for several purposes – for lookup in the text classification JSON file, highlighting in web browser, etc. as described earlier in this document.

**Text as HTML:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN" "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
    <title> From Alice in Wonderland </title>
    <meta name="ilearnrw:words" content="alice.json"/>
    <meta name="dc:title" content="From Alice in Wonderland"/>
    <meta name="dc:creator" content="Lewis Carrol"/>
    <meta name="dc:source" content="Project Gutenberg"/>
    <meta name="dc:subject" content="novel"/>
    <meta name="dc:language" content="en-uk"/>
    <meta name="dc:description" content="Alice in Wonderland by Lewis Carroll is a fictional story about a young girl named Alice who spots an interesting white rabbit, muttering about being late, that she follows down a rabbit hole into a magical world. "/>
  </head>

  <style>h1 {color:black; font-size:40px; font-family:Tahoma, Geneva, sans-serif;}
    h2 {color:black; font-size:35px; font-family:Tahoma, Geneva, sans-serif;}
    h3 {color:black; font-size:30px; font-family:Tahoma, Geneva, sans-serif;}
    p {color:black; font-size:23px; font-family:Tahoma, Geneva, sans-serif;}
  </style>

  </head>
  <body>

<h2 id = p0><sen id = s0><w id = w0>From</w> <w id = w1>Alice</w> <w id = w2>in</w> <w id = w3>Wonderland</w></sen></h2><p id = p1><sen id = s1><w id = w4>Alice</w> <w id = w5>was</w> <w id = w6>not</w> <w id = w7>a</w> <w id = w8>bit</w> <w id = w9>hurt</w>, <w id = w10>and</w> <w id = w11>she</w> <w id = w12>jumped</w> <w id = w13>up</w> <w id = w14>on</w> <w id = w15>to</w> <w id = w16>her</w> <w id = w17>feet</w> <w id = w18>in</w> <w id = w19>a</w> <w id = w20>moment</w>. </sen><sen id = s2><w id = w21>She</w> <w id = w22>looked</w> <w id = w23>up</w>, <w id = w24>but</w> <w id = w25>it</w> <w id = w26>was</w> <w id = w27>all</w> <w id = w28>dark</w> <w id = w29>overhead</w>. </sen><sen id = s3><w id = w30>Before</w> <w id = w31>her</w> <w id = w32>was</w> <w id = w33>another</w> <w id = w34>long</w> <w id = w35>passage</w>, <w id = w36>and</w> <w id = w37>the</w> <w id = w38>White</w> <w id = w39>Rabbit</w> <w id = w40>was</w> <w id = w41>still</w> <w id = w42>in</w> <w id =
```

```
w43>sight</w>, <w id = w44>hurrying</w> <w id = w45>down</w> <w id = w46>it</w>. </sen><sen id = s4><w id = w47>There</w> <w id = w48>was</w> <w id = w49>not</w> <w id = w50>a</w> <w id = w51>moment</w> <w id = w52>to</w> <w id = w53>be</w> <w id = w54>lost</w>. </sen><sen id = s5><w id = w55>Away</w> <w id = w56>went</w> <w id = w57>Alice</w> <w id = w58>like</w> <w id = w59>the</w> <w id = w60>wind</w>, <w id = w61>and</w> <w id = w62>was</w> <w id = w63>just</w> <w id = w64>in</w> <w id = w65>time</w> <w id = w66>to</w> <w id = w67>hear</w> <w id = w68>it</w> <w id = w69>say</w>, <w id = w70>as</w> <w id = w71>it</w> <w id = w72>turned</w> <w id = w73>a</w> <w id = w74>corner</w>, `<w id = w75>Oh</w> <w id = w76>my</w> <w id = w77>ears</w> <w id = w78>and</w> <w id = w79>whiskers</w>, <w id = w80>how</w> <w id = w81>late</w> <w id = w82>it</w>'<w id = w83>s</w> <w id = w84>getting</w>!</sen><sen id = s6><w id = w85>She</w> <w id = w86>was</w> <w id = w87>close</w> <w id = w88>behind</w> <w id = w89>it</w> <w id = w90>when</w> <w id = w91>she</w> <w id = w92>turned</w> <w id = w93>the</w> <w id = w94>corner</w>, <w id = w95>but</w> <w id = w96>the</w> <w id = w97>Rabbit</w> <w id = w98>was</w> <w id = w99>no</w> <w id = w100>longer</w> <w id = w101>to</w> <w id = w102>be</w> <w id = w103>seen</w>. </sen><sen id = s7><w id = w104>She</w> <w id = w105>found</w> <w id = w106>herself</w> <w id = w107>in</w> <w id = w108>a</w> <w id = w109>long</w>, <w id = w110>low</w> <w id = w111>hall</w>, <w id = w112>which</w> <w id = w113>was</w> <w id = w114>lit</w> <w id = w115>up</w> <w id = w116>by</w> <w id = w117>a</w> <w id = w118>row</w> <w id = w119>of</w> <w id = w120>lamps</w> <w id = w121>hanging</w> <w id = w122>from</w> <w id = w123>the</w> <w id = w124>roof</w>.</sen></p>

  </body>
</html>
```

**Text classification JSON file:**

The external JSON file corresponds to the HTML entries and holds the word classification information. This file contains a set of all words in the text document with word text classification within. Each word entry in the JSON file has a corresponding set of ids. We map each word of the HTML file to the word of the JSON file that contains the id of the word. For example, the HTML word with id “w5” corresponds to the JSON word which contains the id 5 in its set of ids.

Below is an example of the same text, however limited to the first word of the document.

```
"wordSet":{
    "idCorrespondance":{
      "0":{
        "word":"from",
        "wordUnmodified":"From",
        "type":"Preposition",
        "syllables":[
```

```
    "from"
  ],
  "phonetics":"frɒm",
  "stem":"from",
  "languageCode":"EN",
  "frequency":1635914.0,
  "graphemesPhonemes":[
    {
      "grapheme":"f",
      "phoneme":"f"
    },
    {
      "grapheme":"r",
      "phoneme":"r"
    },
    {
      "grapheme":"o",
      "phoneme":"ɒ"
    },
    {
      "grapheme":"m",
      "phoneme":"m"
    }
  ],
  "userSeveritiesOnWordProblems":[
    {
      "category":1,
      "index":2,
      "matched":[
        {
          "matchedPart":"o",
          "start":2,
          "end":3
        }
      ],
      "userSeverity":0
    },
    {
      "category":4,
      "index":14,
      "matched":[
        {
```

```
            "matchedPart":"r",
            "start":1,
            "end":2
          }
        ],
        "userSeverity":0
      },
      {
        "category":4,
        "index":16,
        "matched":[
          {
            "matchedPart":"m",
            "start":3,
            "end":4
          }
        ],
        "userSeverity":0
      },
      {
        "category":4,
        "index":22,
        "matched":[
          {
            "matchedPart":"f",
            "start":0,
            "end":1
          }
        ],
        "userSeverity":0
      }
    ],
    "numberOfSyllables":1,
    "length":4
  },
...
```


## 4.6. Web service interface

The Mark-up Component exposes an interface that was built using the Representational State Transfer2 (REST) concept. The API allows receiving data that are JSON or XML encoded messages

via a HTTP POST. Each response of the API is also in JSON format and consists of the HTML document along with the list of the annotated words the HTML document contains.
The API is exposed via a secure channel using standard HTTPS. An authentication token has also to be passed with each call to assure that creation and retrieving of logs is done by an authenticated user.
Also, the API requires a userId that refers to a valid user in order to include his/her severities to each word block it creates. The userId can also be null (i.e. passing no user) to include the case that the reader (which is on the client side) wants to annotate a text without reference to any particular user.
To be able to handle the last case (call of the service with no user) the service also requires the language code of the document. Along with the above the client must provide the text that is about to be marked-up as a body-parameter. After this, the server returns the results in a JSON format.
Each application that wants to have access to the Mark-up Service should call it with the following post method:

```
POST /text/annotate?token=<token>&userId=<user id>&lc=<language code>
```

The above service is public and available under the web address *http://api.ilearnrw.eu/ilearnrw*.
The Mark-up Component would be used by the applications as it is shown on the following sequence diagram of the Figure 7. Sequence Diagram for the Text-Markup ComponentFigure 7.

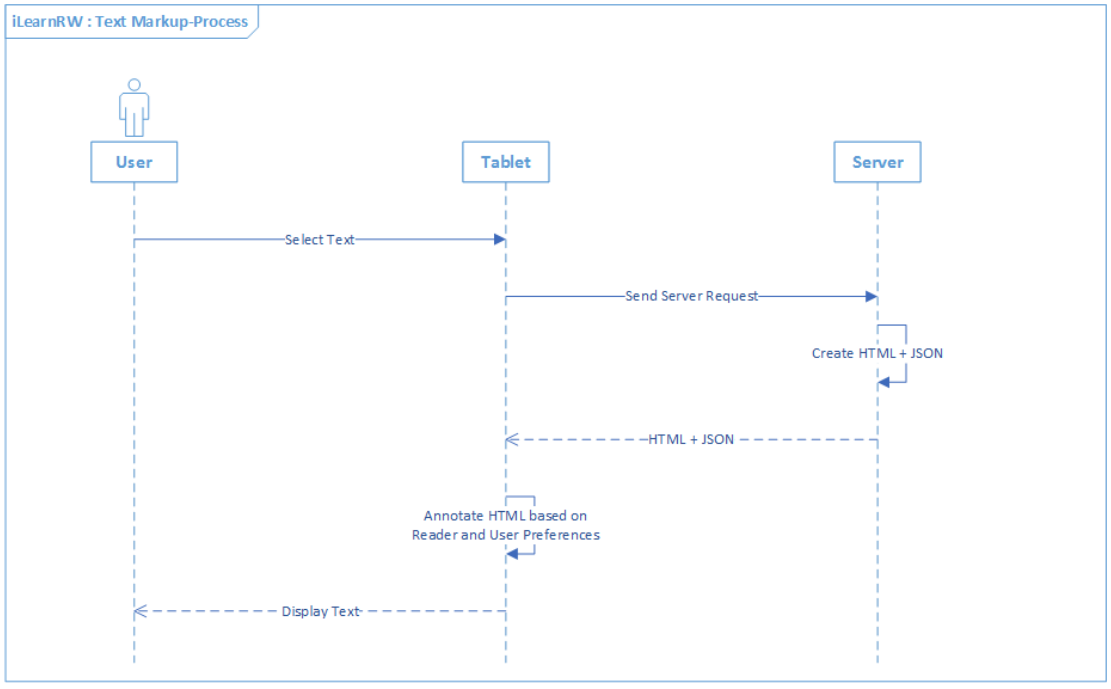


**Figure 7. Sequence Diagram for the Text-Markup Component**

The Server-side components are implemented using the Spring MVC3 framework (Java language). The Spring Web MVC framework is part of the general purpose Spring framework. The Spring MVC framework allows a direct decoupling of the data (the Model), the visual representation (the View) and the logic binding them together (the Controller). We also use the JDBC to access the iLearnRW database since the users' data are stored in MySQL tables.

# 5. Text Adaptation – Text Presentation Modules

So far, we have presented the first step of the text adaptation-presentation process, that is, the call to the web service which annotates a text by processing all the words and discovering potential problem(s). In this section, we describe how the client's side modules of the iLearnRW system interact in order to present a text in a friendly way in the screen of a user.

By this step of the process, the annotated HTML file and the corresponding JSON file have already been generated in the server's side and will be used for the presentation of the text. However, these files provide information about the problem(s) of each word of the text but, no information about how each word will be presented on the screen is included. Consequently, we have to follow a set of mark-up processes that are initiated on the client's side (tablet/ reader application) as part of the CAP API. In brief, a list of such tasks includes the following:

1. Annotation of the HTML file using a set of annotation rules by taking into consideration the user's profile and possible specifications of the user (e.g. the user may prefer to focus on specific problems or has determined specific presentation rules for particular problems). Such rules may be, for instance, "*paint with colour red all "-ing" suffixes"* or "*highlight the parts of the words that match to the problem on which the user currently works with colour green"*.
2. Filter any pages of the HTML file in which there exists a large number of "coloured" words that confuse a child with dyslexia.
3. Present the annotated text on the screen of the user.

The diagram in Figure 8. The Client's Side ProcessFigure 8 presents the client's side processes of the CAP API and consists of three basic modules:

- The *Presentation Rules Module* that is responsible for maintaining and providing the presentation rules for each particular entry of the user's profile. In other words, this module "knows" how each problem will be visualized on the screen (e.g. *for problem "p1.15" paint the problematic parts in colour red*).
- The *Text Annotation Module* that is responsible for (i) communicating with the server, (ii) parsing the HTML files and JSON files generated by the server and (iii) generating an annotated HTML file (by interacting with the Presentation Rules Module) that includes the presentation rules for each word of the text.
- The *Smart Display Module* that displays an annotated HTML file generated either by the Text Annotation Module or manually (e.g. a user may have manually created an annotated HTML file).

In the remainder of this section, we describe in detail each of the above modules and present the methods that the API includes.

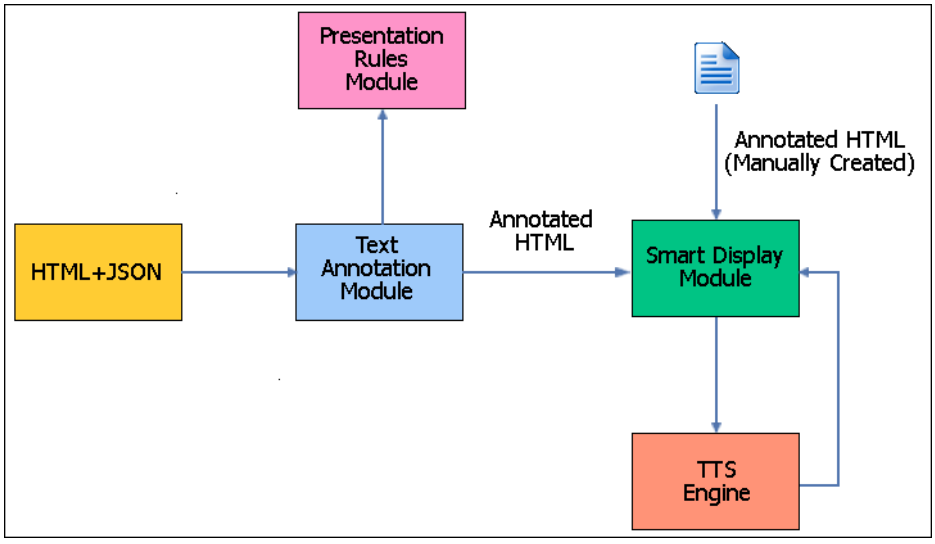


Figure 8. The Client's Side Process

## 5.1. The Presentation Rules Module

The *Presentation Rules Module* is responsible for maintaining and providing the presentation rules for each particular entry of the user's profile. The module can be considered as a two-dimensional table *T[i,j]* (identical to the user's profile table) in which each cell is defined by:

- An *integer* (flag) indicating the presentation rule for the particular problem (e.g. value #1 corresponds to rule "*paint only the problematic part of the word"*).
- A *text colour* that represents the colour that will be used to paint the word or its problematic parts.
- A *highlighting colour* that represents the colour that will be used to highlight the word or its problematic parts.
- A *boolean* value indicating whether the corresponding problem is activated (for the purposes of the presentation) or not.

The initialization of the Presentation Rules Module is performed based on the child's profile and the specifications of the user. For instance, consider the case where a user prefers to work only on the "-*ing*" suffix difficulty. In order to achieve this, the Presentation Rules Module should be initialized (or call an appropriate method if the module has already been initialized) such that only the cell that corresponds to the "-*ing*" suffix difficulty is activated. The initialization requires a user's profile and the language code of the document (to include the case where no reference to a particular user is given). Also, the API supports the modification of the presentation rule (including both the rule itself but, also the corresponding colours) for a particular problem. The default presentation rules about each problem can be defined by a reader application.

In the following, we present the methods that the *Presentation Rules Module* API should provide:

- **void initializePresentationRules(Profile profile, String languageCode)**
  - *Input:* A user profile and the document's language code.
  - *Output:* ---
  - *Description:* Initializes a Presentation Rules object.

- **void setUserProfile (Profile profile)**
  - *Input:* A user profile.
  - *Output:* ---
  - *Description:* Sets a new user profile to the Presentation Rules object.

- **Profile getUserProfile ()**
  - *Input:* ---
  - *Output:* Returns the current user profile.

- **void setLanguageCode (String languageCode)**
  - *Input:* A new language code
  - *Output:* ---
  - *Description:* Sets a language code to the Presentation Rules object.

- **String getLanguageCode ()**
  - *Input:* ---
  - *Output:* Returns the current language code.

- **void setPresentationRule(int i, int j, int r)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry and an integer *r* indicating the desired presentation rule.
  - *Output:* ---
  - *Description:* Sets presentation rule *r* to the profile entry that corresponds to indices *i,j*.

- **int getPresentationRule(int i, int j)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry.
  - *Output:* Returns an integer indicating the presentation rule for the profile entry that corresponds to indices *i,j*.

- **void setTextColor(int i, int j, Color color)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry and a colour representing the desired text colour.
  - *Output:* ---
  - *Description:* Sets the desired text colour for the problem that corresponds to indices *i,j* of the user's profile table.

- **Color getTextColor(int i, int j)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry.
  - *Output:* Returns the text colour for the problem that corresponds to indices *i,j* of the user's profile table.

- **void setHighlightingColor(int i, int j, Color color)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry and a colour corresponding to the desired highlighting colour.
  - *Output:* ---

- *Description:* Sets the highlighting colour for the problem that corresponds to indices *i,j* of the user's profile table.

- **Color getHighlightingColor(int i, int j)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry.
  - *Output:* Returns the highlighting colour for the problem that corresponds to indices *i,j* of the user's profile table.

- **void setActivated(int i, int j, boolean flag)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry and a boolean flag indicating whether the particular problem is activated.
  - *Output:* ---
  - *Description:* Sets whether or not the problem that corresponds to indices *i,j* of the user's profile table is activated or not.

- **boolean getActivated (int i, int j)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry.
  - *Output:* Returns true if the problem that corresponds to indices *i,j* of the user's profile is activated, false otherwise.

## 5.2. The Text Annotation Module

The *Text Annotation Module* is the basic module of the CAP's architecture that communicates and exchanges information with the remaining modules of the system. The Text Annotation Module takes as input the text file that has to be presented and the profile of the child. Again to cover the case where there is no reference to a particular user, the module also requires the language code of the document. The basic tasks of this module are the following:

1. **Sends the text file to the Text Adaptation Web Service and gets the HTML and the JSON files generated by the server.**

2. **Parses the HTML file and splits it in *pages* based on the dimensions of the screen that will be used for the presentation**. In this step, spans with tags <page> and </page> that determine the pages are added within the HTML document such that each spanned area fits to the screen of a tablet with the specified dimensions.

3. **Parses the JSON file**. In this step, a new java object is created, referred to as WordClassifierInfo that includes all the information contained in the JSON file. This object will be used during the processing of each word of the text.

4. **For each word of a page, it calculates its difficulty based on the user's profile and calls the Presentation Rules Module to check whether the word will be annotated and how it will be presented**. To achieve this, the Text Annotation Module has to read the word from the text, calculate its difficulty and identify the problem into which a particular word falls (based on the WordClassifierInfo object generated by the JSON file). Then, it calls the Presentation Rules Module to specify if and how this problem is decorated. In the case where the particular

problem is not activated in the table of the Presentation Rules Module then, the word will not be decorated. Otherwise, the Presentation Rules Module should return the corresponding presentation rule. Having determined the presentation rule for a particular problem, the module adds appropriate CSS properties to the HTML file that capture the desired presentation rule (e.g. Alice, etc.).

5. **Filters a page.** Since, it is possible that many difficult words that fall into the same problem or linguistic category may appear in the same page, the resulting presentation may become inappropriate for a child with dyslexia (e.g. many highlighted or colored words in the text). For this reason, this module should filter the problematic pages based on thresholds defined by the user within a reader application. There exist diverse approaches regarding the filtering process. We have decided to define thresholds based on the percentage of words that are decorated in the same page. By default, we set this threshold to 30% which implies that only 30% of the words of a page will be eventually decorated. One issue that arises is how to select difficult words given that there may exist several occurrences of a word in the same page. For this reason, this module orders the words based on their severity values as determined in the JSON file and decorates only their first occurrence in the page. This would allow the annotation of a greater number of difficult words in the same page. However, a potential user of the API can overwrite the corresponding filtering method as she wishes.

6. **Returns an appropriately annotated HTML file.** This file contains all the necessary information and presentation rules that will be given as input to the *Smart Display Module*.

In the following, we present the methods that the Text Annotator Module API should provide:

- **void initializeTextAnnotator(String textFile, Profile profile, String languageCode)**
  - *Input:* The name of a text file, the user's profile and the language code of the document.
  - *Output:* ---
  - *Description:* Initializes a TextAnnotator object.

- **void setTextFile(String textFile)**
  - *Input:* The name of a text file.
  - *Output:* ---
  - *Description:* Sets a text file to the TextAnnotator object.

- **String getTextFile ()**
  - *Input:* ---
  - *Output:* Returns the text file of the TextAnnotator object.

- **void setProfile(Profile profile)**
  - *Input:* A user's profile.
  - *Output:* ---
  - *Description:* Sets a user's profile to the TextAnnotator object.

- **Profile getProfile ()**
  - *Input:* ---
  - *Output:* Returns the current user's profile of the TextAnnotator object.

- **void setLanguageCode(String languageCode)**
  - *Input:* The language code of the document.
  - *Output:* ---
  - *Description:* Sets the language code to the TextAnnotator object.

- **String getLanguageCode ()**
  - *Input:* ---
  - *Output:* Returns the language code of the TextAnnotator object.

- **void setJSONFile(String JSONFileName)**
  - *Input:* The name of the JSON file.
  - *Output:* ---
  - *Description:* Sets a JSON file to the TextAnnotator object.

- **String getJSONFile()**
  - *Input:* ---
  - *Output:* Returns the JSON file of the TextAnnotator.

- **void setInputHTMLFile(String HTMLFile)**
  - *Input:* The name of an input HTML file.
  - *Output:* ---
  - *Description:* Sets an HTML file to the TextAnnotator object.

- **String getInputHTMLFile()**
  - *Input:* ---
  - *Output:* Returns the input HTML file of the TextAnnotator.

- **void annotateText()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Annotates the HTML file associated with the TextAnnotator object.

- **WordClassifierInfo parseJSONFile()**
  - *Input:* ---
  - *Output:* Returns a java object that includes all the information of the JSON file.

- **Map<Integer, List<String>> splitInPages()**
  - *Input:* ---
  - *Output:* Returns a Map object maintaining a list of words of each page (including multiple occurrences of each word in the same page).

- *Description:* Besides than returning a map, it also spans with tags <page> and </page> that determine the pages are added within the HTML document such that each spanned area fits to the screen of a tablet with the specified dimensions.

- **String readNextWord()**
  - *Input:* ---
  - *Output:* Returns the next word that has to be processed in the HTML file.

- **double calculateWordDifficulty(String word)**
  - *Input:* An input word
  - *Output:* Returns the difficulty of the input word.

- **void setWordColor(int wordId, Color color, int start, int end)**
  - *Input:* A word id, a colour in HEX format (e.g. w1, #FF6699), an integer *start* indicating the starting index of the problematic part, and an integer *end* indicating the finishing index of the problematic part of a word.
  - *Output:* ---
  - *Description:* The method applies the CSS "color" property to the word with id=w1, starting from index *start* and finishing to index *end* and annotates appropriately the HTML file.

- **void removeWordColor(int wordId)**
  - *Input:* A word id, an integer *start* indicating the starting index of the problematic part, and an integer *end* indicating the finishing index of the problematic part of a word.
  - *Output:* ---
  - *Description:* Removes the colour of a word by removing the CSS "color" property and annotates appropriately the HTML file.

- **void setWordHighlighting(int wordId, Color color, int start, int end)**
  - *Input:* A word id, a colour in HEX format (e.g. w1, #FF6699), an integer *start* indicating the starting index of the problematic part, and an integer *end* indicating the finishing index of the problematic part of a word.
  - *Output:* ---
  - *Description:* The method applies the CSS "mark" property to the word with id=w1, starting from index *start* and finishing to index *end* and annotates appropriately the HTML file.

- **void removeWordHighlighting(int wordId)**
  - *Input:* A word id, an integer *start* indicating the starting index of the problematic part and an integer *end* indicating the finishing index of the problematic part of a word.
  - *Output:* ---
  - *Description:* Removes the highlight colour of a word by removing the CSS "mark" property and annotates appropriately the HTML file.

- **void setHighlightingSpeed(int wordId, double value)**

- *Input:* A word id and a fraction from 0.1 to 2 (e.g. "faster", 1.5)
- *Output:* ---
- *Description:* The method adds a tag to the word with the specified id value that determines the speed of the highlighting when it will display the word (if highlighting operation is activated) and annotates appropriately the HTML file.

- **List<String> decoratedWordsPerPage (int p)**
  - *Input:* An integer *p* indicating a particular page.
  - *Output:* Returns a List object containing the decorated words of page *p*.

- **void setThreshold(double d)**
  - *Input:* A fraction *p* between 0 and 1 indicating that the decorated words will be no more than *d%* of the words within the same page.
  - *Output:* ---

- **double getThreshold()**
  - *Input:* ---
  - *Output:* Returns a fraction between 0 and 1 indicating that only the *d*% of the words of a page will be decorated.

- **void filterPage(int p)**
  - *Input:* An integer *p* indicating the page on which the filtering will apply.
  - *Output:*---
  - *Description:* Filters the page such that only the *d*% of the words of a page will be decorated.

- **void setTextFontFamily(Font font)**
  - *Input:* A font face value.
  - *Output:* ---
  - *Description:* Modifies the font face of the HTML file by applying the CSS "font-family" property.

- **Font getTextFontFamily()**
  - *Input:* ---
  - *Output:* Returns the current font family of the HTML file.

- **void setTextFontSize(double v)**
  - *Input:* A font face value *v* between 0.8 and 2.
  - *Output:* ---
  - *Description:* Modifies by *v%* the font size of the HTML file by applying the CSS "font-size" property and annotates appropriately the HTML file.

- **double getTextFontSize()**
  - *Input:* ---
  - *Output:* Returns the current font-size value (as percentage) of the HTML file.

- **void setTextFontSize(int v)**
  - *Input:* A font face value *v* in pixels (px).
  - *Output:* ---
  - *Description:* Sets to the font size of the HTML file, value v (in pixels) by applying the CSS "font-size" property and annotates appropriately the HTML file.

- **int getTextFontSize()**
  - *Input:* ---
  - *Output:* Returns the current font-size value (in pixels) of the HTML file.

- **void setLineSpacing(double v)**
  - *Input:* A value *v* representing the line spacing.
  - *Output:* ---
  - *Description:* Modifies by *v%* the line spacing size of the HTML file by applying the CSS "line-height" property and annotates appropriately the HTML file

- **double getLineSpacing ()**
  - *Input:* ---
  - *Output:* Returns the current value of line spacing (as percentage) of the HTML file.

- **void setLineSpacing(int v)**
  - *Input:* A value *v* representing the line spacing.
  - *Output:* ---
  - *Description:* Sets to the line spacing size of the HTML file value v (in pixel) by applying the CSS "line-height" property and annotates appropriately the HTML file.

- **int getLineSpacing ()**
  - *Input:* ---
  - *Output:* Returns the current value of line spacing (in pixel) of the HTML file.

- **void setMargin(double v)**
  - *Input:* A value *v* representing the margins of the HTML file.
  - *Output:* ---
  - *Description:* Modifies by *v%* the margins of the HTML file by applying the CSS "margin" property and annotates appropriately the HTML file.

- **double getMargin()**
  - *Input:* ---
  - *Output:* Returns the current value of margins (as percentage) of the HTML file.

- **void setMargin(int v)**
  - *Input:* A value *v* representing the margins of the HTML file.
  - *Output:* ---

- *Description:* Sets to the margins of the HTML file value v (in pixels) by applying the CSS "margin" property and annotates appropriately the HTML file.

- **int getMargin()**
  - *Input:* ---
  - *Output:* Returns the current value of margins (in pixels) of the HTML file.

- **void setBackgroundColor(Color color)**
  - *Input:* A colour in HEX format
  - *Output:* ---
  - *Description:* Sets the background colour of the HTML file by applying the "background-color" property and annotates appropriately the HTML file.

- **Color getBackgroundColor()**
  - *Input:* ---
  - *Output:* Returns the current background colour of the HTML file.

- **void setForegroundColor(Color color)**
  - *Input:* A colour in HEX format
  - *Output:* ---
  - *Description:* Sets the background colour of the HTML file by applying the "color" property and annotates appropriately the HTML file.

- **Color getForegroundColor()**
  - *Input:* ---
  - *Output:* Returns the current foreground colour of the HTML file.

- **void updatePageStyle(String CSSProperty, String value)**
  - *Input:* A CSS property and a value (e.g. color, "red")
  - *Output:* ---
  - *Description:* Applies the CCS style to the HTML document.

Methods of this module which are of general interest (e.g. finding difficulty of a word, parsing of JSON file, etc.) will be made available to other packages.

## 5.3. Smart Display Module

In this subsection, we describe the basic functionality of the *Smart Display Module* that is responsible for displaying annotated HTML files. Basically, the Smart Display Module consists of a presentation panel and a set of buttons that control operations such as highlighting, navigation, TTS operation, etc. The presentation panel is based on the WebView java class (android.webkit.WebView) which implements the basic functionalities in order to present HTML pages. The Smart Display Module is able to display any appropriately annotated HTML file that is either generated by the Text Annotation Module or has been created manually by a user (in this case, the user can also provide a JSON file).

In the following, we present the methods that the Smart Display Module API provides:

- **void initializeSmartDisplay(String HTMLFileName)**
  - *Input:* The name of the HTML file.
  - *Output:* ---
  - *Description:* Initializes a smart display object for the specific HTML file.

- **String getCurrentDocument ()**
  - *Input:* ---
  - *Output:* Returns the name of the HTML file that is currently presented.

- **void setCurrentDocument (String fileName)**
  - *Input:* The name of an HTML file.
  - *Ouput:* ---
  - *Description:* Updates the smart display object with a new HTML.

- **void setJSONFile(String JSONFileName)**
  - *Input:* The name of the JSON file.
  - *Output:* ---
  - *Description:* Sets a JSON file to the TextAnnotator object.

- **String getJSONFile()**
  - *Input:* ---
  - *Output:* Returns the JSON file of the TextAnnotator.

- **boolean isHighlighting()**
  - *Input:* ---
  - *Output:* Returns true if highlighting operation is activated.

- **boolean onLoadFinished ()**
  - *Input:* ---
  - *Ouput:* ---
  - *Description:* Notifies when that a file has finished loading.

- **void startHighlighting()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Starts the highlighting operation from the current cursor position.

- **void stopHighlighting()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Stops the highlighting operation.

- **void pauseHighlighting()**
  - *Input:* ---
  - *Output:* ---

- *Description:* Pauses the highlighting operation.

- **void enableHighlighting()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Enables the highlighting operation

- **void disableHighlighting()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Disables the highlighting operation.

- **void setHighlightingSpeed(double k)**
  - *Input:* A fraction *k* between 0.1 and 2.
  - *Output:* ---
  - *Description:* Sets the highlighting speed to *k*% of the normal speed. The changes won't apply if the TTS is enabled. In that case the colour highlighter will follow the TTS.

- **double getHighlightingSpeed()**
  - *Input:* ---
  - *Output:* Returns the current highlighting speed.

- **void gotoDocumentBeginning()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Navigates to the beginning of the document.

- **boolean existsDocumentPageNumber(int pageNumber)**
  - *Input:* A page number.
  - *Output:* Returns true if the page exists in the document.

- **void setCurrentPage(int pageNumber)**
  - *Input:* A page number
  - *Output:* ---
  - *Description:* Sets the display to the k-th page of the text.

- **int getNumberOfPages()**
  - *Input:* ---
  - *Output:* Returns the total number of pages.

- **List<String>getSelectedWords()**
  - *Input:* ---
  - *Output:* Returns a list of the selected words of the page.

- **void gotoNextScreen()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Navigates to the next screen.

- **boolean existsNextScreen()**
  - *Input:* ---
  - *Output:* Returns true if there exists a next page in the document.

- **boolean existsPreviousScreen()**
  - *Input:* ---
  - *Output:* Returns true if there exists a previous page in the document.

- **void gotoPreviousScreen()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Navigates to the previous screen.

- **boolean existsLargerFontSize()**
  - *Input:* ---
  - *Output:* Returns true if there exists a larger font size.

- **void increaseFontSize()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Increases font size.

- **boolean existsSmallerFontSize()**
  - *Input:* ---
  - *Output:* Returns true if there exists a smaller font size.

- **void decreaseFontSize()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Decreases font size.

- **Color getBackgroundColor()**
  - *Input:* ---
  - *Output:* Returns current background colour.

- **void setBackgroundColor(Color color)**
  - *Input:* A background colour.
  - *Output:* ---
  - *Description:* Sets the desired background colour.

- **void addPresentationListener(PresentationListener listener)**
  - *Input:* A PresentationListener object.
  - *Output:* ---
  - *Description:* Adds a PresentationListener that implements "on touch" events to the SmartDisplay object.

- **void removeContentPresentationListener (PresenterListener listener)**
  - *Input:* A PresentationListener object.
  - *Output:* ---
  - *Description:* Removes a PresentationListener from the SmartDisplay object.

- **List<Integer> search(String s)**
  - *Input:* A word/sentence *s* to be searched.
  - *Output:* Returns a list of pages containing *s*.

- **void search(String s)**
  - *Input:* A word/sentence *s* to be searched.
  - *Output:* ---
  - *Description:* Marks the appearances of String *s* within the current page.

- **List<Integer>searchInPage(String s, int p)**
  - *Input:* A word/sentence *s* and a page number *p*.
  - *Output:* Returns a list with the positions on which *s* was found within page *p*.

- **void setCursor(int k)**
  - *Input:* An integer *k*.
  - *Output:* Sets the cursor to the k-thword displayed on the screen.

- **void startTTS()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Starts the TTS from the current cursor position with the highlighter (if is enabled).

- **void setTTSSpeed(double k)**
  - *Input:* A double value *k* within range 0.1 and 2
  - *Output:* Sets the speed of the TTS voice to *k*% of the normal speed.

- **int stopTTS()**
  - *Input:* ---
  - *Output:* Stops the TTS and returns the current cursor position.

- **int pauseTTS()**
  - *Input:* ---
  - *Output:* Stops the TTS and returns the current cursor position.

- **void speakSentence(int id)**
  - *Input:* The id of a sentence.
  - *Output:* ---
  - *Description:* Starts the TTS with input the sentence with the given id.

- **void speakWord(int id)**
  - *Input:* The id of a word.
  - *Output:* ---
  - *Description:* Starts the TTS with input the word with the given id.

- **void onTouchEvent(Event e)**
  - *Input:* A touch event *e* on the screen
  - *Output:* Returns the word under the touch point.

- **Button getStartHighlightingButton()**
  - *Input:* ---
  - *Output:* Returns the button that starts the highlighting.

- **Button getStopHighlightingButton()**
  - *Input:* ---
  - *Output:* Returns the button that stops the highlighting.

- **Button getPauseHighlightingButton()**
  - *Input:* ---
  - Output: Returns the button that pauses the highlighting.

- **TonggleButton getActivateHighlightingButton()**
  - *Input:* ---
  - *Output:* Returns the button that activates/deactivates the highlighting

- **Button getPreviousPageButton()**
  - *Input:* ---
  - *Output:* Returns the button that navigates to the previous page.

- **Button getNextPageButton()**
  - *Input:* ---
  - *Output:* Returns the button that navigates to the next page.

- **Button getStartTTSButton()**
  - *Input:* ---
  - *Output:* Returns the button that starts the TTS engine.

- **Button getStopTTSButton()**
  - *Input:* ---

- *Output:* Returns the button that stops the TTS engine.

- **Button getPauseTTSButton()**
  - *Input:* ---
  - Output: Returns the button that pauses the TTS engine.

- **Button getFastForwardTTSButton()**
  - *Input:* ---
  - Output: Returns the button that navigates the TTS engine to the next sentence.

- **Button getRewindTTSButton()**
  - *Input:* ---
  - Output: Returns the button that navigates the TTS engine to the previous sentence.

- **SeekBar getVolumeTTSSeekBar ()**
  - *Input:* ---
  - Output: Returns the SeekBar object that controls the volume of the TTS engine.

- **SeekBar getSpeedTTSSeekBar ()**
  - *Input:* ---
  - Output: Returns the s SeekBar lider object that controls the speed of the TTS engine.

- **SeekBar getPitchTTSSeekBar ()**
  - *Input:* ---
  - Output: Returns the SeekBar object that controls the pitch of the TTS engine.

- **TonggleButton getActivateTTSButton()**
  - *Input:* ---
  - *Output:* Returns the button that activates/deactivates the TTS engine.

# 6. A Demonstration of the Usage of the CAP API on a Web-Application

In this section, we present an example demonstrating how the CAP API can be used in a web-application. For the demonstration, we have selected one user that encounters language problems in the English language and one user that encounters language problems in the Greek language. For each of these users, their profile will be taken under consideration and the system will present the text in such a way that the particular problems that each user is facing will be identified and presented in an appropriate manner.

**Use-case 1:** User A is having difficulties in English words that include:

1. Letter "t" which is pronounced as "t". For example: stress's, timed, stressing.
2. Syllable "tt" which is pronounced as "t". For example: cutting, potted, matter's.
3. Suffix "ed" which adding to the word causes no changes. For example: schooled, chickened, clouded.

The system took under consideration the above problems and the result is presented in Figure 9. In words in which problem 1 applies, letter "t" is presented in colour-red. Words that contain syllable "tt" are highlighted with colour-green, while in words that fall into the third problem-type, only suffix "ed" is coloured in blue.

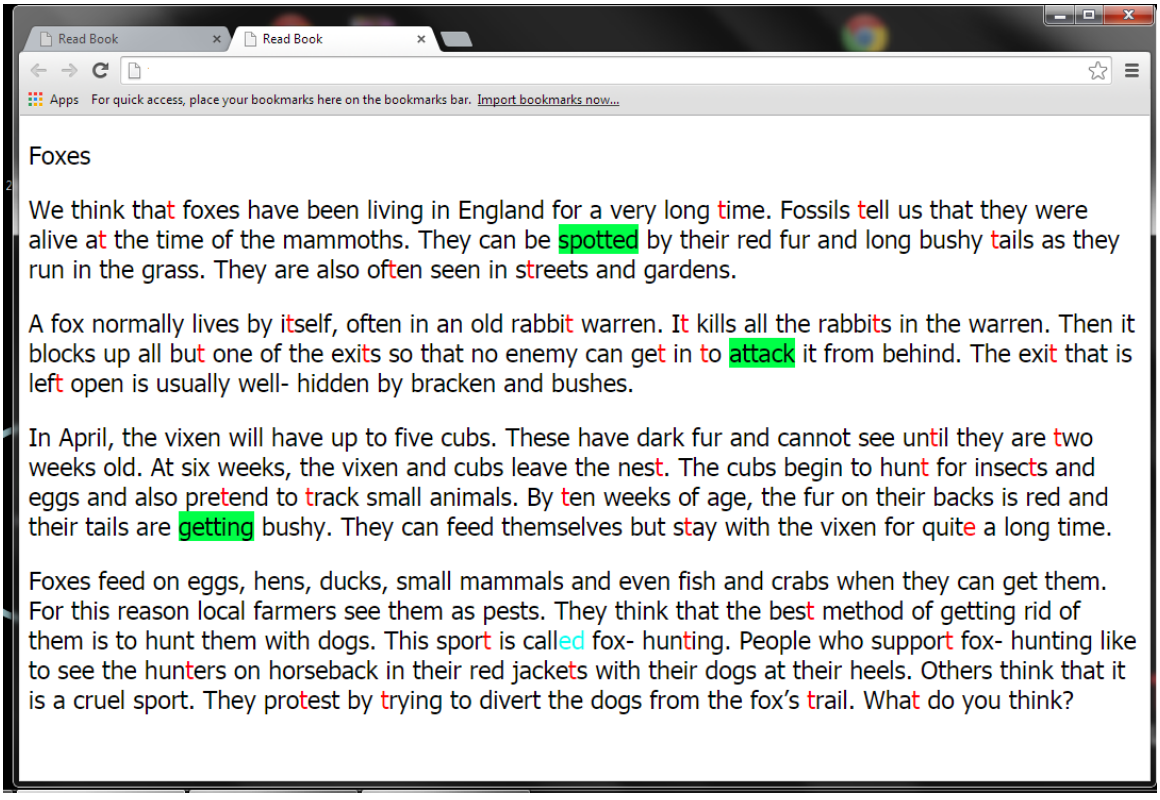


**Figure 9. English text presentation example.**

In Figure 9 we present the result of an English text presentation for a user facing problems with (i) letter "t" when is pronounced as "t", (ii) Syllable "tt" which is pronounced as "t" and (iii) Suffix

"ed" which adding to the word causes no changes. Figure 10 shows the result of a Greek text presentation for a user facing problems with (i) consonants, like "μ" and "ν" that sound similar, (ii) Consonants like "φ, θ, Θ, Β, β" that are visually similar and (iii) words that contain the pattern "cv-v".

**Use-case 2:** User B is having difficulties in Greek words that include:

1. Consonants, like "μ" and "ν" that sound similar.
2. Consonants like "φ, θ, Θ, Β, β" that are visually similar.
3. Words that contain the pattern "cv-v"

The system took under consideration the above problems and the result is presented in Figure 10. Words in which problem 1 applies are coloured in red. In words that fall into the second category, only problematic letters are distinguished and coloured in yellow, while words that include the problematic pattern of problem 3 are coloured in light-blue.

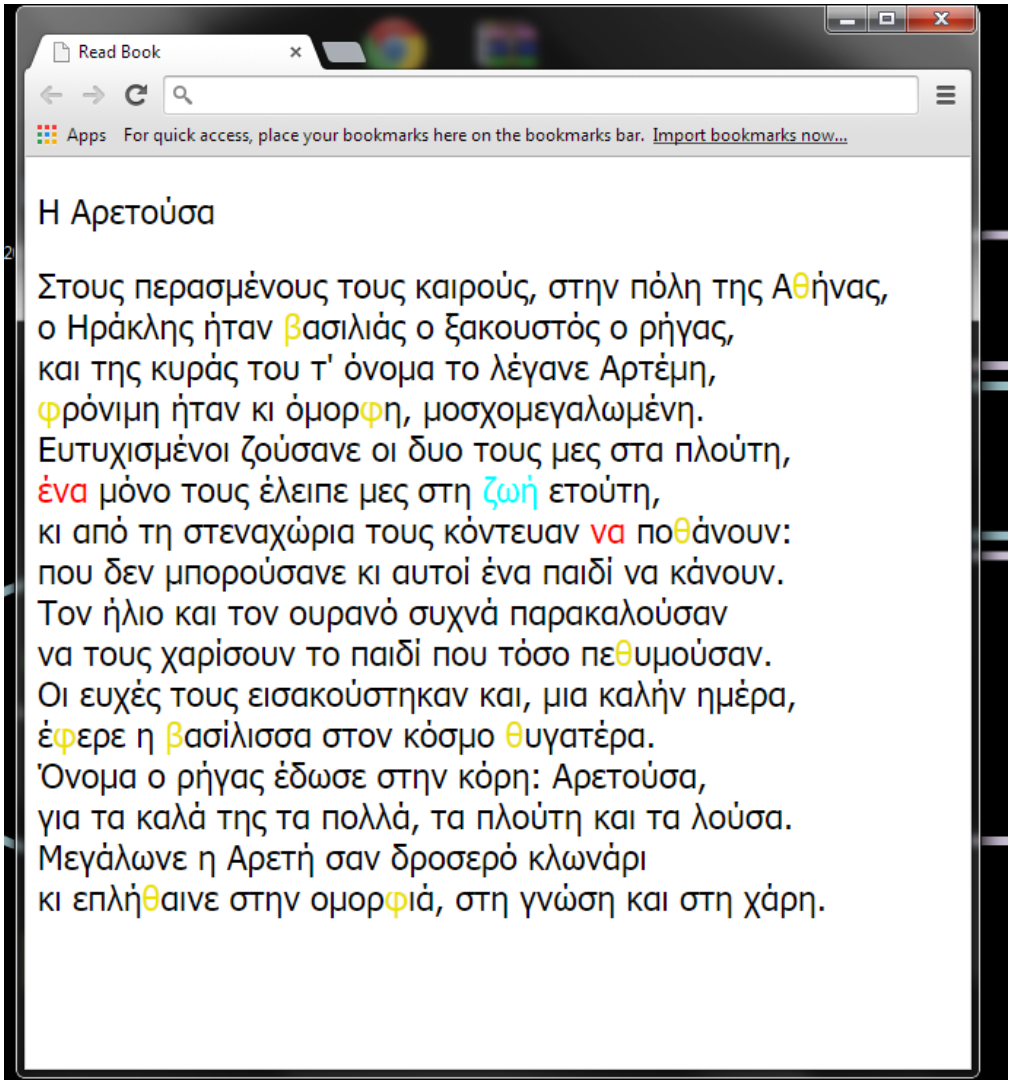


**Figure 10. The result of a Greek text example.**

# 7. Updated Version of the CAP API

During the implementation of a Reader Client, we encountered some difficulties which required minor and insignificant modifications to the designed API. For these reason, we provide below the methods that were modified in the different modules of the CAP API.

## 7.1. The Updated Presentation Rules Module

In the Presentation Rules Module, we performed two minor changes. The first concerns the languageCode field of each user (which determines the language in which the user encounters difficulties). This field was associated with the user's profile and thus, it was not necessary to be given as an input to the Presentation Rules Module object. Thus, method **initializePresentationRules()** was simplified and methods **setLanguageCode()** and **getLanguageCode()** were removed. More precisely:

- void initializePresentationRules(Profile profile)
  - *Input:* A user profile.
  - *Output:* ---
  - *Description:* Initializes a Presentation Rules object.

  --- **Simplified:** Instead of **initializePresentationRules (Profile profile, String languageCode)**

- void setLanguageCode (String languageCode)
  - *Input:* A new language code
  - *Output:* ---
  - *Description:* Sets a language code to the Presentation Rules object.

  --- **Removed**

- String getLanguageCode ()
  - *Input:* ---
  - *Output:* Returns the current language code.

  --- **Removed**

The second modification concerns the java-object that represents the field "Color". We decided to modify it from Color to int, since in android applications, it is easier to work with the integer values of the colours. Thus, the following methods were modified

- **void setTextColor(int i, int j, int color)**
  - *Input:* A pair of indices *i*,*j* that refers to a particular profile entry and an integer value representing the desired text colour.
  - *Output:* ---
  - *Description:* Sets the desired text colour for the problem that corresponds to indices *i*,*j* of the user's profile table.

  --- **Modified:** Instead of **void setTextColor(int i, int j, Color color)**

- **int getTextColor(int i, int j)**

- *Input:* A pair of indices *i,j* that refers to a particular profile entry.
- *Output:* Returns the text colour for the problem that corresponds to indices *i,j* of the user's profile table.

--- **Modified:** Instead of **Color getTextColor(int i, int j)**

- **void setHighlightingColor(int i, int j, int color)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry and an integer corresponding to the desired highlighting colour.
  - *Output:* ---
  - *Description:* Sets the highlighting colour for the problem that corresponds to indices *i,j* of the user's profile table.

  --- **Modified:** Instead of **void setHighlightingColor(int i, int j, Color color)**

- **int getHighlightingColor(int i, int j)**
  - *Input:* A pair of indices *i,j* that refers to a particular profile entry.
  - *Output:* Returns the highlighting colour for the problem that corresponds to indices *i,j* of the user's profile table.

  --- **Modified:** Instead of **Color getHighlightingColor(int i, int j)**

## 7.2. The Updated Text Annotation Module

In this module, there have been several minor modifications to the existing methods of the API (some because of the two modifications in the languageCode and Color fields as described above), while some new methods were added. In the following, we present all methods that were modified and mention the changes that have been applied.

- **void initializeTextAnnotator(String textFile, Profile profile)**
  - *Input:* The name of a text file and the user's profile.
  - *Output:* ---
  - *Description:* Initializes a TextAnnotator object.

  --- **Simplified:** Instead of **void initializeTextAnnotator(String textFile, Profile profile, String languageCode)**

- **void initializePresentationModule (Profile profile)**
  - *Input:* A user profile.
  - *Output:* ---
  - *Description:* Initializes a Presentation Rules object.

  --- **Added**

- **void initializePresentationModuleFromServer(String token, String userID)**
  - *Input:* The authentication token and the user ID.
  - *Output:* ---
  - *Description:* Initializes the presentation module based in the profile of the user requested from the server.

--- **Added**

- **void setLanguageCode(String languageCode)**
  - *Input:* The language code of the document.
  - *Output:* ---
  - *Description:* Sets the language code to the TextAnnotator object.

  --- **Removed**

- **String getLanguageCode ()**
  - *Input:* ---
  - *Output:* Returns the language code of the TextAnnotator object.

  --- **Removed**

- **void setJSON(String inputJSON)**
  - *Input:* The name of the input JSON.
  - *Output:* ---
  - *Description:* Sets a JSON input to the TextAnnotator object.

  --- **Renamed:** Instead of **void setJSONFile(String JSONFileName)**

- **String getJSONInput()**
  - *Input:* ---
  - *Output:* Returns the JSON file of the TextAnnotator.

  --- **Renamed:** Instead of **String getJSONFile ()**

- **void setAnnotatedHTML(String annotatedHTML)**
  - *Input:* The name of the annotated HTML.
  - *Output:* ---
  - *Description:* Sets the annotated HTML to the TextAnnotator object.

  --- **Added**

- **String getAnnotatedHTML()**
  - *Input:* ---
  - *Output:* Returns the input HTML of the TextAnnotator.

  --- **Added**

- **UserBasedAnnotatedWordsSet retrieveJSonObject()**
  - *Input:* ---
  - *Output:* ---
  - Retrieves the java object from the JSON file as a UserBasedAnnotatedWordsSet object.

  --- **Renamed:** Instead of **WordClassifierInfo parseJSONFile().**

- **void activatePagination(boolean activatePagination)**
  - *Input:* Activates the text pagination.
  - *Output:* True if pagination is activated, false otherwise.

--- **Added**

- **void setWordColor(String wordId, int color, int start, int end)**
  - *Input:* A word id, the integer value of a colour, an integer *start* indicating the starting index of the problematic part, and an integer *end* indicating the finishing index of the problematic part of a word.
  - *Output:* ---
  - *Description:* The method applies the CSS "color" property to the word with id=w1, starting from index *start* and finishing to index *end* and annotates property the HTML file.

  --- **Modified:** Instead of **void setWordColor(int wordId, Color color, int start, int end)**

- **void removeWordColor(String wordID, int start, int end)**
  - *Input:* A word id, an integer *start* indicating the starting index of the problematic part, and an integer *end* indicating the finishing index of the problematic part of a word.
  - *Output:* ---
  - *Description:* Removes the colour of a word by removing the CSS "color" property and annotates property the HTML file.

  --- **Modified:** Instead of **void removeWordColor(int wordId)**

- **void setWordHighlighting(String wordId, int color, int start, int end)**
  - *Input:* A word id, the integer value of a colour, an integer *start* indicating the starting index of the problematic part, and an integer *end* indicating the finishing index of the problematic part of a word.
  - *Output:* ---
  - *Description:* The method applies the CSS "mark" property to the word with id=w1, starting from index *start* and finishing to index *end* and annotates property the HTML file.

  --- **Modified:** Instead of **void setWordHighlighting(int wordId, Color color, int start, int end)**

- **void removeWordHighlighting(String wordID, int start, int end)**
  - *Input:* A word id, an integer *start* indicating the starting index of the problematic part and an integer *end* indicating the finishing index of the problematic part of a word.
  - *Output:* ---
  - *Description:* Removes the highlight colour of a word by removing the CSS "mark" property and annotates property the HTML file.

  --- **Modified:** Instead of **void removeWordHighlighting(int wordId)**

- **void setHighlightingSpeed(String wordId, double value)**
  - *Input:* A word id and a fraction from 0.1 to 2 (e.g. "faster", 1.5)
  - *Output:* ---
  - *Description:* The method adds a tag to the word with the specified id value that determines the speed of the highlighting when it will display the word (if highlighting operation is activated) and annotates property the HTML file.

--- **Modified:** Instead of **void setHighlightingSpeed(int wordId, double value)**

- **void setAttributesToWord(String wordID, java.util.Map<String, String> attributes, int start, int end)**
  - *Input:* A word id, a *map* with attributes and their corresponding values, an integer *start* indicating the starting index of the problematic part, and an integer *end* indicating the finishing index of the problematic part of a word.
  - *Output:*---
  - *Description:* Sets a list of attributes to the word with a specific ID.

  --- **Added**

- **void setTextFontFamily(String font, String tag)**
  - *Input:* A font family value and the tag that will be affected.
  - *Output:* ---
  - *Description:* Modifies the font face of the HTML file by applying the CSS "font-family" property.

  --- **Modified:** Instead of **void setTextFontFamily(Font font)**

- **String getTextFontFamily()**
  - *Input:* ---
  - *Output:* Returns the current font family of the HTML file.

  --- **Modified:** Instead of **Font getTextFontFamily()**

- **void setTextFontSize(double v, String tag))**
  - *Input:* A font face value *v* between 0.8 and 2 and the tag that will be affected.
  - *Output:* ---
  - *Description:* Modifies by *v%* the font size of the HTML file by applying the CSS "font-size" property and annotates property the HTML file.

  --- **Modified:** Instead of **void setTextFontSize(double v)**

- **void setTextFontSize(int v, String tag))**
  - *Input:* A font face value *v* in pixels (px) and the tag that will be affected.
  - *Output:* ---
  - *Description:* Sets to the font size of the HTML file, value v (in pixels) by applying the CSS "font-size" property and annotates property the HTML file.

  --- **Modified:** Instead of **void setTextFontSize(int v)**

- **void setLineSpacing(double v, String tag))**
  - *Input:* A value *v* representing the line spacing and the tag that will be affected.
  - *Output:* ---
  - *Description:* Modifies by *v%* the line spacing size of the HTML file by applying the CSS "line-height" property and annotates property the HTML file.

  --- **Modified:** Instead of **void setLineSpacing(double v)**

- **void setLineSpacing(int v, String tag))**
  - *Input:* A value *v* representing the line spacing and the tag that will be affected.
  - *Output:* ---
  - *Description:* Sets to the line spacing size of the HTML file value v (in pixel) by applying the CSS "line-height" property and annotates property the HTML file.

  --- **Modified:** Instead of **void setLineSpacing(int v)**

- **void setMargin(double v, String tag)**
  - *Input:* A value *v* representing the margins of the HTML file and the tag that will be affected.
  - *Output:* ---
  - *Description:* Modifies by *v%* the margins of the HTML file by applying the CSS "margin" property and annotates property the HTML file.

  --- **Modified:** Instead of **void setMargin(double v)**

- **void setMargin(int v, String tag)**
  - *Input:* A value *v* representing the margins of the HTML file and the tag that will be affected.
  - *Output:* ---
  - *Description:* Sets to the margins of the HTML file value v (in pixels) by applying the CSS "margin" property and annotates property the HTML file

  --- **Modified:** Instead of **void setMargin(int v)**

- **void setBackgroundColor(int color, String tag)**
  - *Input:* The integer value of colour and the tag that will be affected
  - *Output:* ---
  - *Description:* Sets the background colour of the HTML file by applying the "background-color" property and annotates property the HTML file.

  --- **Modified:** Instead of **void setBackgroundColor(Color color)**

- **int getBackgroundColor()**
  - *Input:* ---
  - *Output:* Returns the current background colour of the HTML file.

  --- **Modified:** Instead of **Color getBackgroundColor()**

- **void setForegroundColor(int color, String tag)**
  - *Input:* The integer value of colour and the tag that will be affected.
  - *Output:* ---
  - *Description:* Sets the background colour of the HTML file by applying the "color" property and annotates property the HTML file.

  --- **Modified:** Instead of **void setForegroundColor(Color color)**

- **int getForegroundColor()**
  - *Input:* ---

- *Output:* Returns the current foreground colour of the HTML file.

--- **Modified:** Instead of **Color getForegroundColor ()**

- **void updatePageStyle(String attribute, String value, String tag)**
  - *Input:* A CSS property, a value (e.g. color, "red") and the tag that will be affected
  - *Output:* ---
  - *Description:* Applies the CCS style to the HTML document.

--- **Modified:** Instead of **updatePageStyle(String CSSProperty, String value)**

## 7.3. The Updated Smart Display Module

In this section, we present the modification performed in the Smart Display Module.

- **int getBackgroundColor()**
  - *Input:* ---
  - *Output:* Returns current background colour.

--- **Modified:** Instead of **Color getBackgroundColor()**

- **void setBackgroundColor(int color)**
  - *Input:* The integer value of a color representing the background colour.
  - *Output:* ---
  - *Description:* Sets the desired background colour.

--- **Modified:** Instead of **void setBackgroundColor(Color color)**

- **void addOnClickListener(OnClickListener listener)**
  - *Input:* An OnClickListener object.
  - *Output:* ---
  - *Description:* Adds an OnClickListener that implements "on click" events to the SmartDisplay object.

--- **Added**

- **void removeOnClickListener (OnClickListener listener)**
  - *Input:* An OnClickListener object
  - *Output:* ---
  - *Description:* Removes an OnClickListener from the SmartDisplay object.

--- **Added**

- **void addOnLongClickListener(OnLongClickListener listener)**
  - *Input:* An OnLongClickListener object.
  - *Output:* ---
  - *Description:* Adds an OnLongClickListener that implements "on long click" events to the SmartDisplay object.

--- **Added**

- **void removeOnLongClickListener (OnLongClickListener listener)**
  - *Input:* An OnLongClickListener object
  - *Output:* ---
  - *Description:* Removes an OnLongClickListener from the SmartDisplay object.

  --- **Added**

- **void addOnSeekBarChangeListener (OnSeekBarChangeListener listener)**
  - *Input:* An OnSeekBarChangeListener object.
  - *Output:* ---
  - *Description:* Adds an OnSeekBarChangeListener that implements events related to seek bar changes to the SmartDisplay object.

  --- **Added**

- **void removeOnSeekBarChangeListener (OnSeekBarChangeListener listener)**
  - *Input:* An OnSeekBarChangeListener object
  - *Output:* ---
  - *Description:* Removes an OnSeekBarChangeListener from the SmartDisplay object.

  --- **Added**

- **void setTTS (TTS tts)**
  - *Input:* An TTS object
  - *Output:* ---
  - *Description:* Adds a TTS-object to the SmartDisplay object.

  --- **Added**

- **void removeSearches()**
  - *Input:* ---
  - *Output:* ---
  - *Description:* Removes all current searches.

  --- **Added**

- **boolean isReadingFinished ()**
  - *Input:* ---
  - *Output:* Returns false if highlighting and/or TTS are still running, true otherwise.
  - *Description:* Returns false if highlighting and/or TTS are still running, true otherwise.

  --- **Added**

# 8. Conclusions

In this deliverable, we presented the Content Adaptation and Presentation packages, in the form of application programming interfaces (APIs), that aim to present a text on the screen appropriately annotated such that it is suitable for a specific user. For the text presentation, the system takes into consideration potential difficulties from the profile of the user and follows the presentation rules that are defined for each profile entry problem.

NOTE: This is an updated version of deliverable 4.2 which was submitted in month 18. It includes a revised implementation of the CAP API (section 7) and an example demonstrating its usage (section 6) in a web application.