Visit Notes Analysis Module Developer's Guide

Kavya Katipally, Ryan Eshleman
Advisor: Dr. Barry Levine

TABLE OF CONTENTS

1. Overview

The Visit Notes Analysis module was built to provide Named Entity Recognition (NER) capabilities to OpenMRS to help the system extract more information from Visit Note text as well as other potential sources of unstructured text.

Part of the challenge of enhancing OpenMRS's ability to analyze plain text is to help the user gain the most value out of the analysis. Our module provides one use, summarizing and visualizing the information in a patient note, however we understand that there are many possible uses of NER.

It is with this understanding that we write the Developer's Guide in order to provide future developers with the background necessary to either continue and refine the development of this module, or leverage the NER API provided by this module in order to build new and better projects.

In this guide you will find five sections:

1. **System Design**: A high level description of the three main components working under the hood to carry out the module's NER functions.
2. **Application Programming Interface**: Details and examples of the main Java classes used to support the NER functionality.
3. **REST web services:** Details of new REST end points created for this module.
4. **User Interface development:** AngularJS has been used in the front end. Some details regarding the front end code are shared in this section.
5. **Package Structure**: A diagram of the package structure with brief descriptions of each package.

2. System Design

There are three major system design components that a developer should be familiar with in order to continue development with this module. Those components are:

1. The SofaDocument data model
2. The Named Entity Recognition algorithm
3. Processing Visit Notes with Aspect Oriented Programming.

2.1 SofaDocument data model

The module adds four database tables to the OpenMRS data model corresponding to the SofaDocument data hierarchy ('sofa' is shorthand for **S**ubject **of A**nalysis). The data stored in these tables is accessible via the service layer through the service class NLPService. The data model is shown in figure 1.

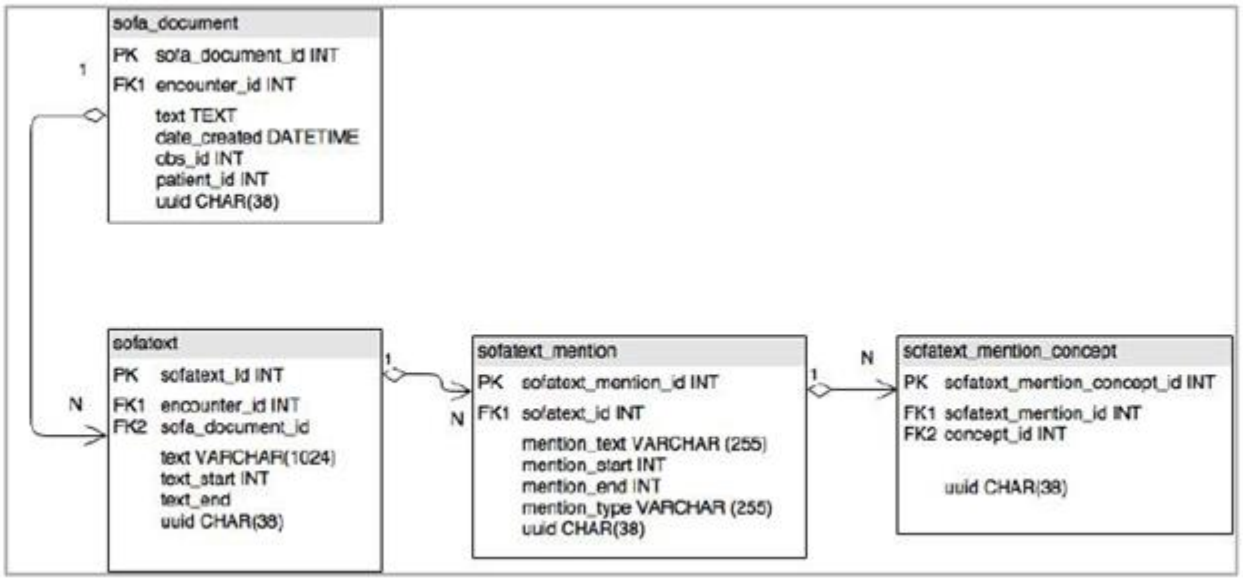


Figure 1: SofaDocument data model

This data model forms a hierarchical structure used to maintain the results of the analysis of a Visit Note. Roughly the tables form this correspondence:

sofa_document	=> Visit Note
sofatext	=> sentence in Visit Note
sofatext_mention	=> entity identified in a sentence
sofatext_mention_concept	=> corresponding OpenMRS Concept for the entity, if it exists. Empty otherwise.

Leveraging the NLPService class and Hibernate ORM mappings (www.hibernate.org), we can read sofa_documents from the database into Java Objects. The corresponding data hierarchy of Java Objects is shown in figure 2.

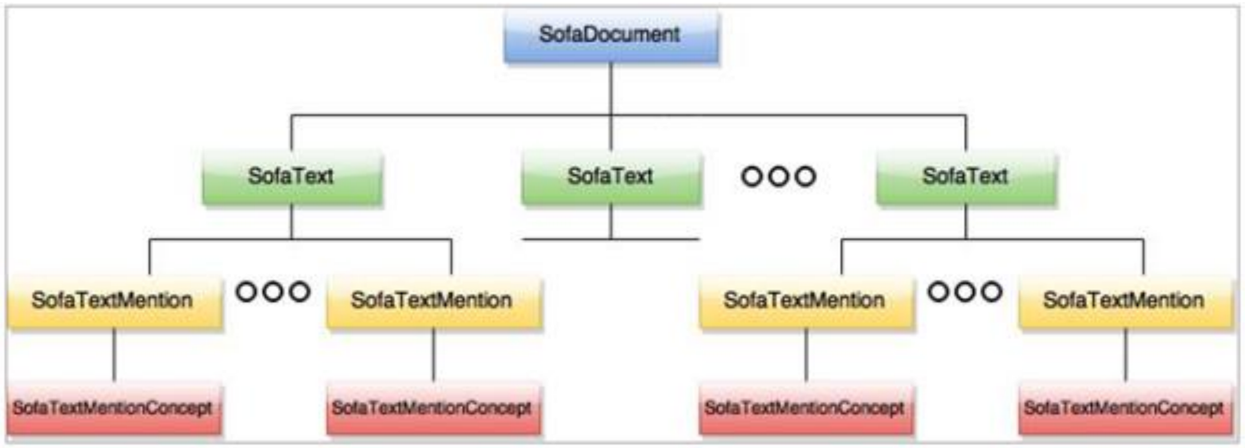


Figure 2: SofaDocument data hierarchy

One SofaDocument can contain many SofaText objects. One SofaText can contain many SofaTextMention objects. One SofaTextMention can contain a SofaTextMentionConcept object.

A concrete example of this hierarchy is shown in figure 3.

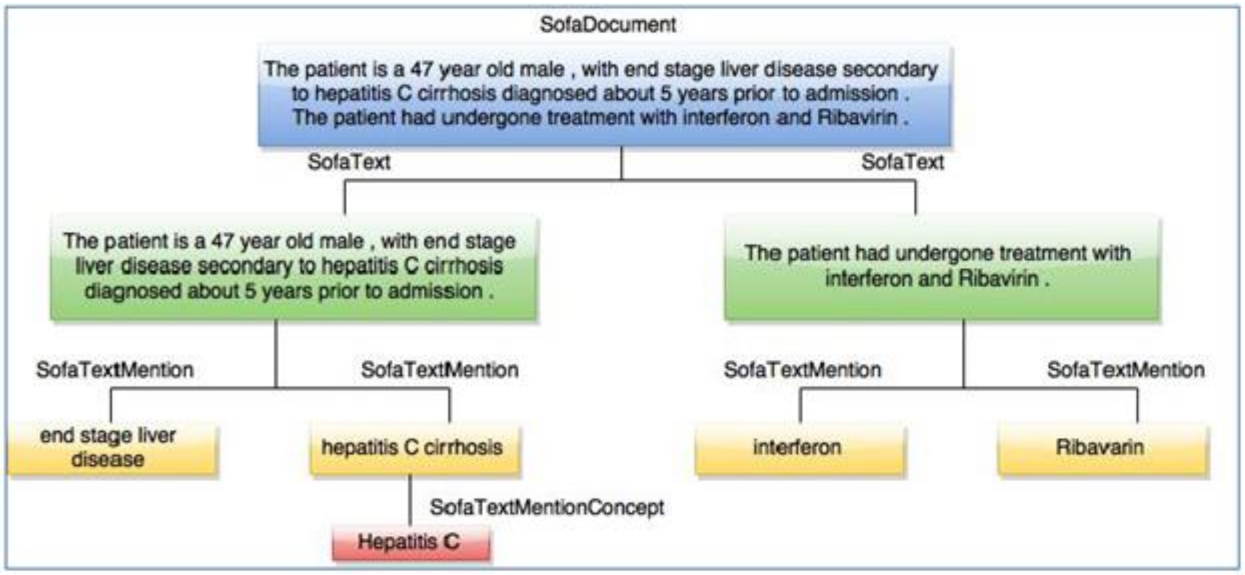

SofaDocument
The patient is a 47 year old male , with end stage liver disease secondary to hepatitis C cirrhosis diagnosed about 5 years prior to admission . The patient had undergone treatment with interferon and Ribavirin .
SofaText
SofaText
The patient is a 47 year old male , with end stage liver disease secondary to hepatitis C cirrhosis diagnosed about 5 years prior to admission .
The patient had undergone treatment with interferon and Ribavirin .
SofaTextMention
SofaTextMention
SofaTextMention
SofaTextMention
end stage liver disease
hepatitis C cirrhosis
interferon
Ribavarin
SofaTextMentionConcept
Hepatitis C

Figure 3: concrete example of SofaDocument data hierarchy

2.2 Named Entity Recognition Algorithm and Implementation

The Named Entity Recognition algorithm takes place in two steps. The first step uses the concept class mappings described in section 3.1 to identify concepts and their synonyms within the text. The second step uses the machine learning algorithm Conditional Random Fields provided by BANNER (banner.sourceforge.com) to find entities that may not be explicitly noted in the Concept Dictionary. Figure 4 diagrams this process. BANNER was chosen for this module after it showed superior performance when compared with several other open source NER systems.

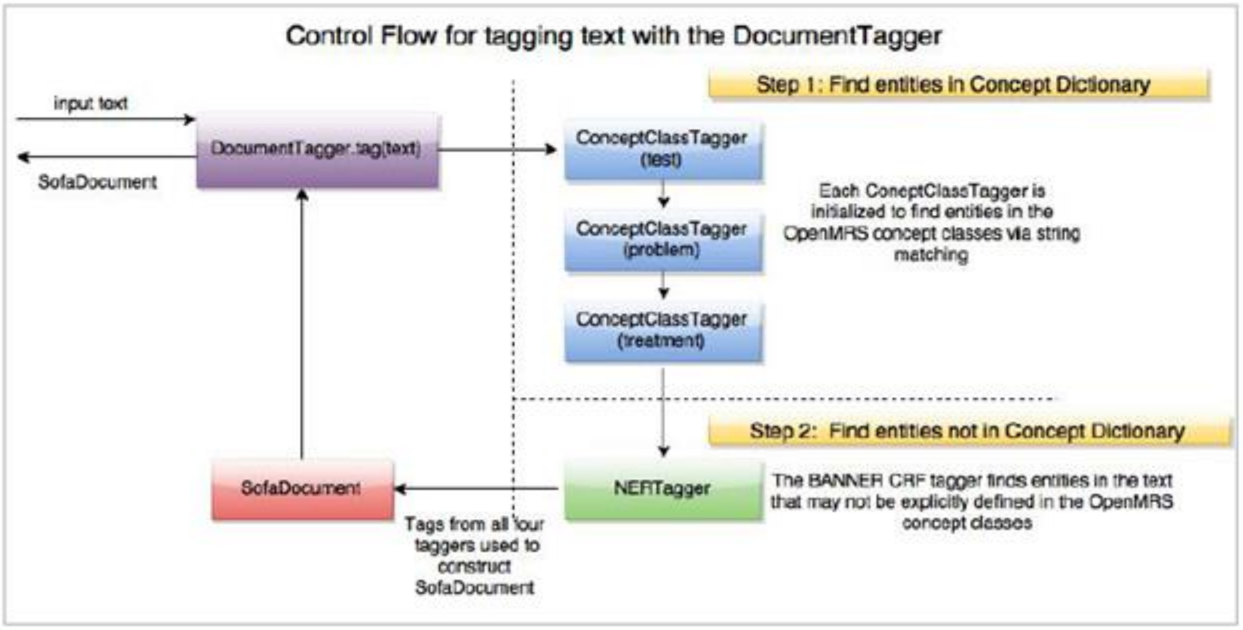

Control Flow for tagging text with the DocumentTagger
input text
DocumentTagger.tag(text)
SofaDocument
Step 1: Find entities in Concept Dictionary
ConceptClassTagger (test)
ConceptClassTagger (problem)
ConceptClassTagger (treatment)
Each ConeptClassTagger is initialized to find entities in the OpenMRS concept classes via string matching
Step 2: Find entities not in Concept Dictionary
NERTagger
The BANNER CRF tagger finds entities in the text that may not be explicitly defined in the OpenMRS concept classes
SofaDocument
Tags from all four taggers used to construct SofaDocument

Figure 4: Named Entity Recognition Algorithm

2.3 Visit Note Processing with Aspect Oriented Programming

Visit notes are analyzed as they are submitted to the system via Spring's support for Aspect Oriented Programming (https://wiki.openmrs.org/display/docs/OpenMRS+AOP). When a user submits a visit note through the Visit Notes page, the VisitNoteAdvice class interrupts the control flow and processes the text of the Visit Note, saving the results for future presentation. Figure 5 shows this process.

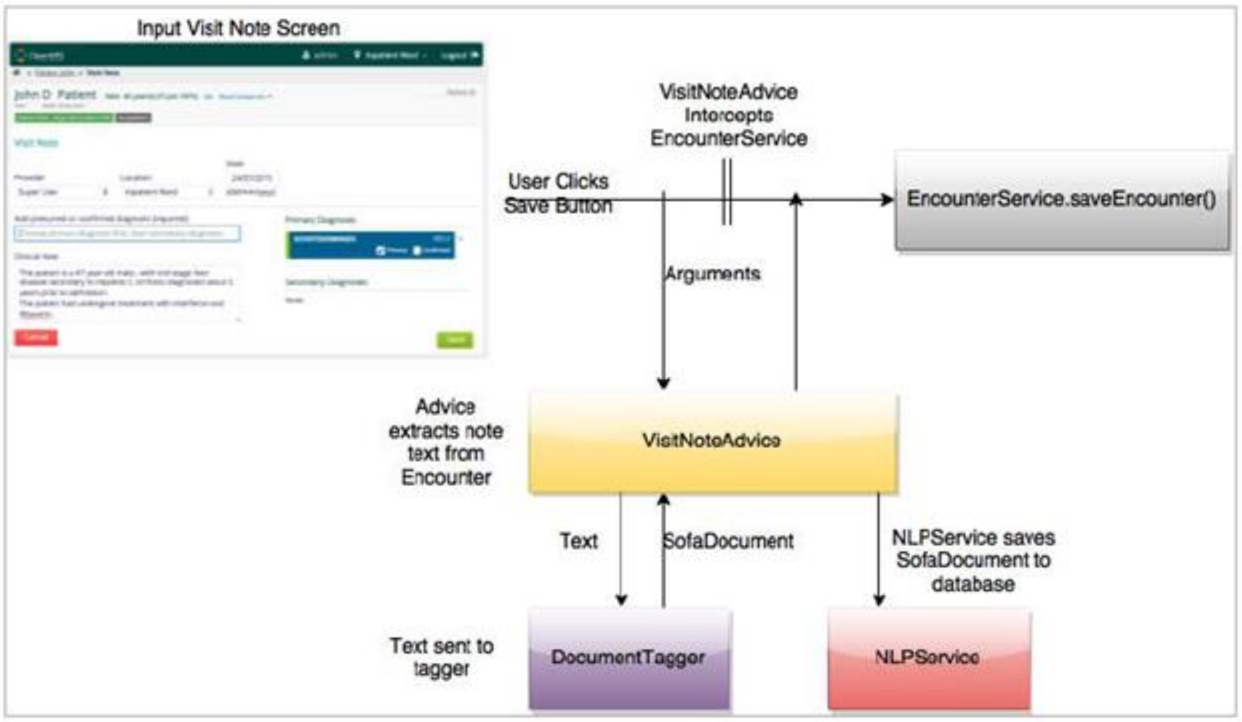

Input Visit Note Screen
VisitNoteAdvice Intercepts EncounterService
User Clicks Save Button
EncounterService.saveEncounter()
Arguments
Advice extracts note text from Encounter
VisitNoteAdvice
Text
SofaDocument
NLPService saves SofaDocument to database
Text sent to tagger
DocumentTagger
NLPService

Figure 5: AOP processing of visit notes

These three components - the data model, the algorithm, and the control flow of capturing a Visit Note make up the foundation of the module.

3. Application Programming Interface

There are three main components of the API a developer can use to extend or build on top of this module. Those components are the NLPService class for loading/storing data objects, the SofaDocument and related classes for manipulating the data and the DocumentTagger class for performing NER on strings of text. Below are brief descriptions of each component followed by illustrative code snippets. Finally we provide excerpts from the JavaDocs for the related classes.

NLPService class: This service manages storing and retrieving SofaDocument objects and related data in the database.

SofaDocument, SofaText, SofaTextMention and SofaTextMentionConcept classes:

Objects of these classes are used to manipulate the text data and annotations.

DocumentTagger class: This class provides a simple interface to the developer to execute the NER functionality of the module.

3.1 Code Snippets

The following are four code snippets that illustrate how to use the core API functions.

1. Using the DocumentTagger class to annotate a text string and print out all “problem” entities found in that string:

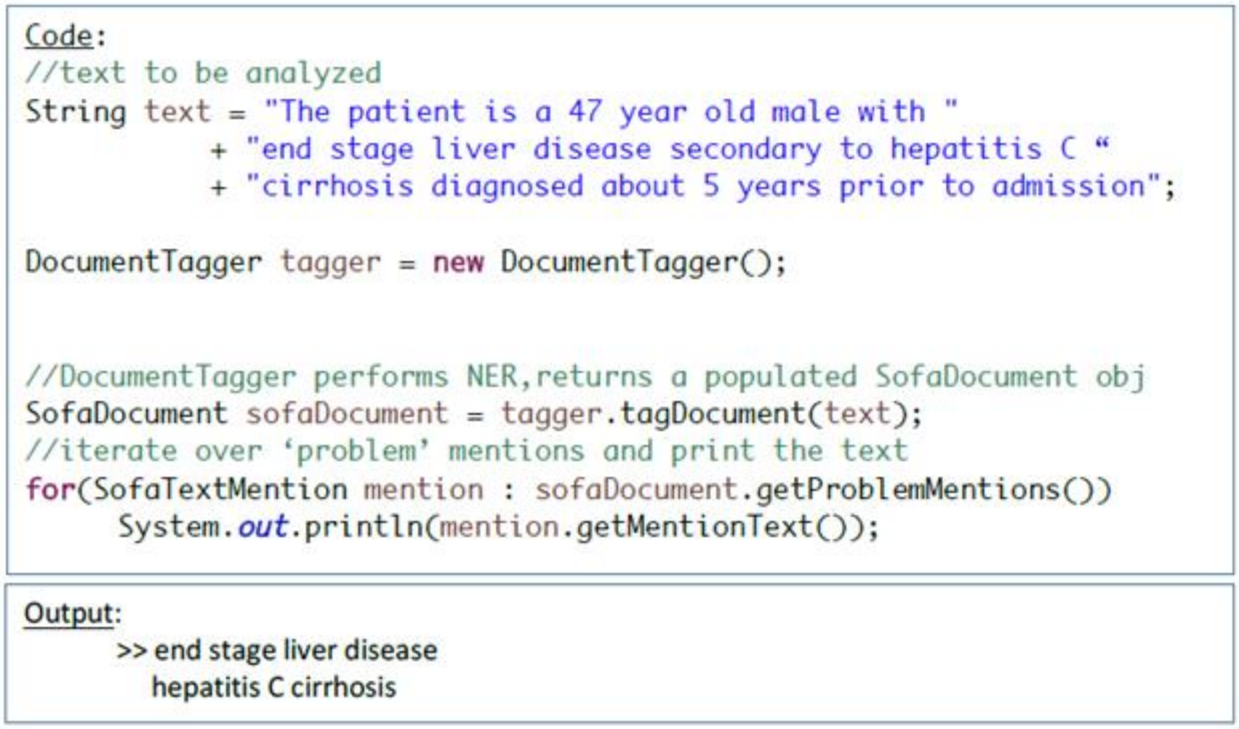

Code:
```
//text to be analyzed
String text = "The patient is a 47 year old male with "
          + "end stage liver disease secondary to hepatitis C “
          + "cirrhosis diagnosed about 5 years prior to admission";

DocumentTagger tagger = new DocumentTagger();


//DocumentTagger performs NER,returns a populated SofaDocument obj
SofaDocument sofaDocument = tagger.tagDocument(text);
//iterate over ‘problem’ mentions and print the text
for(SofaTextMention mention : sofaDocument.getProblemMentions())
     System.out.println(mention.getMentionText());
```
Output:
```
     >> end stage liver disease
        hepatitis C cirrhosis
```

Figure 6: Tag text with DocumentTagger

2. Tag a string of text and save it to the database using the NLPService class.

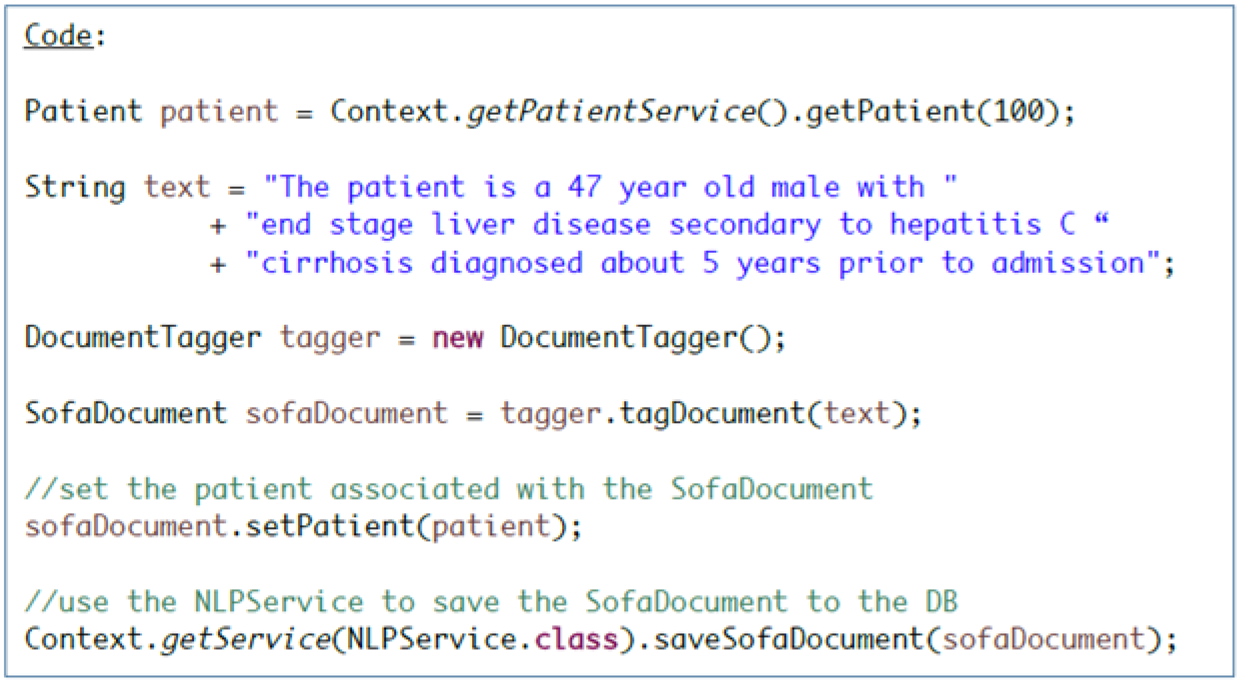
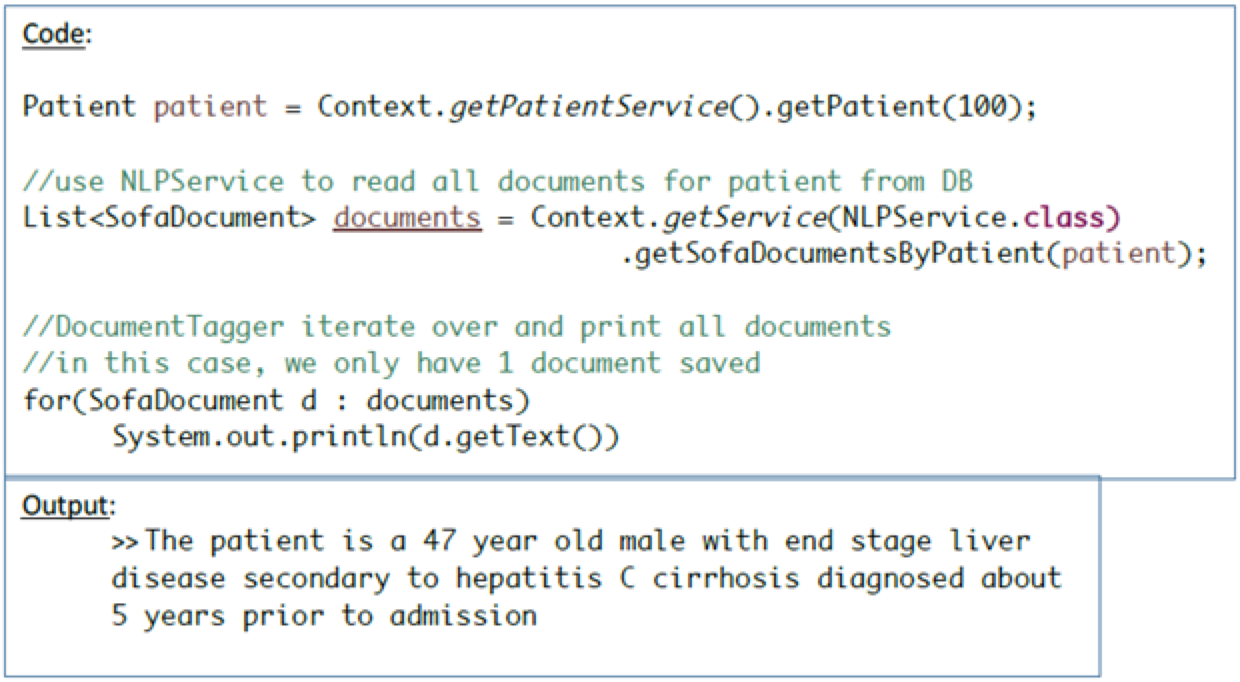

Code:
```
Patient patient = Context.getPatientService().getPatient(100);

String text = "The patient is a 47 year old male with "
          + "end stage liver disease secondary to hepatitis C “
          + "cirrhosis diagnosed about 5 years prior to admission";

DocumentTagger tagger = new DocumentTagger();

SofaDocument sofaDocument = tagger.tagDocument(text);

//set the patient associated with the SofaDocument
sofaDocument.setPatient(patient);

//use the NLPService to save the SofaDocument to the DB
Context.getService(NLPService.class).saveSofaDocument(sofaDocument);
```

Figure 7: Tag a string, then save it to the database using NLPService

3. Retrieve all SofaDocuments for patient whose patient_id is 100, and print out the text of the documents.



<u>Code</u>:

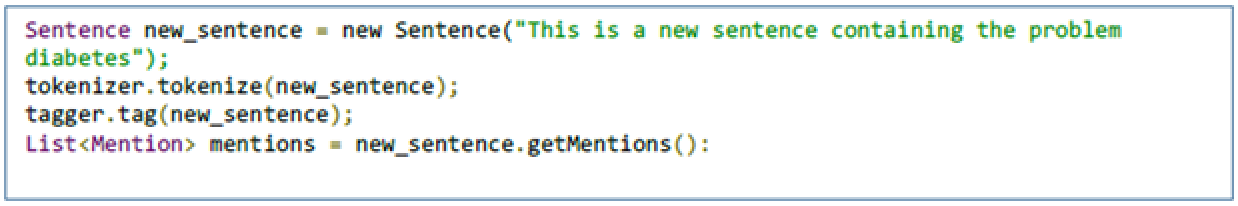
```
Patient patient = Context.getPatientService().getPatient(100);

//use NLPService to read all documents for patient from DB
List<SofaDocument> documents = Context.getService(NLPService.class)
                                       .getSofaDocumentsByPatient(patient);

//DocumentTagger iterate over and print all documents
//in this case, we only have 1 document saved
for(SofaDocument d : documents)
      System.out.println(d.getText())
```

<u>Output</u>:

```
      >>The patient is a 47 year old male with end stage liver
      disease secondary to hepatitis C cirrhosis diagnosed about
      5 years prior to admission
```

Figure 8: Retrieve SofaDocument objects associated with a patient, then print the text

4. Internally to the DocumentTagger object, code from the BANNER library is being executed to contribute annotations to the text. Here, tagger is an instance of the BANNER CRFTagger class, tokenizer is an instance of the BANNER Tokenizer class, Sentence and Mention are also classes from the BANNER library. Figure 9 gives this example of the CRFTagger tagging a sentence, the result is a list of Mention objects.



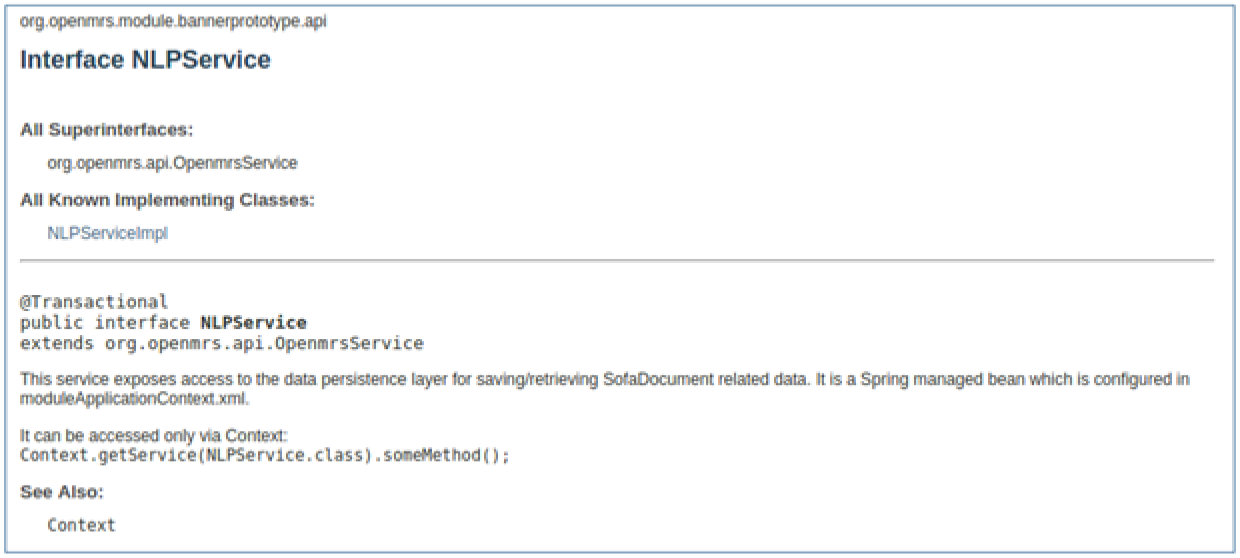
```
Sentence new_sentence = new Sentence("This is a new sentence containing the problem
diabetes");
tokenizer.tokenize(new_sentence);
tagger.tag(new_sentence);
List<Mention> mentions = new_sentence.getMentions():
```

Figure 9: example of BANNER library code

3.2 Javadoc Excerpts

Note: Many of these classes reference a banner.tagging.Mention object. This object is part of the result of the NER tagging performed by BANNER. It contains information like mention text and mention type that are used in many of the following classes.

3.2.1 NLPService



org.openmrs.module.bannerprototype.api

Interface NLPService

All Superinterfaces:

org.openmrs.api.OpenmrsService

All Known Implementing Classes:

NLPServiceImpl

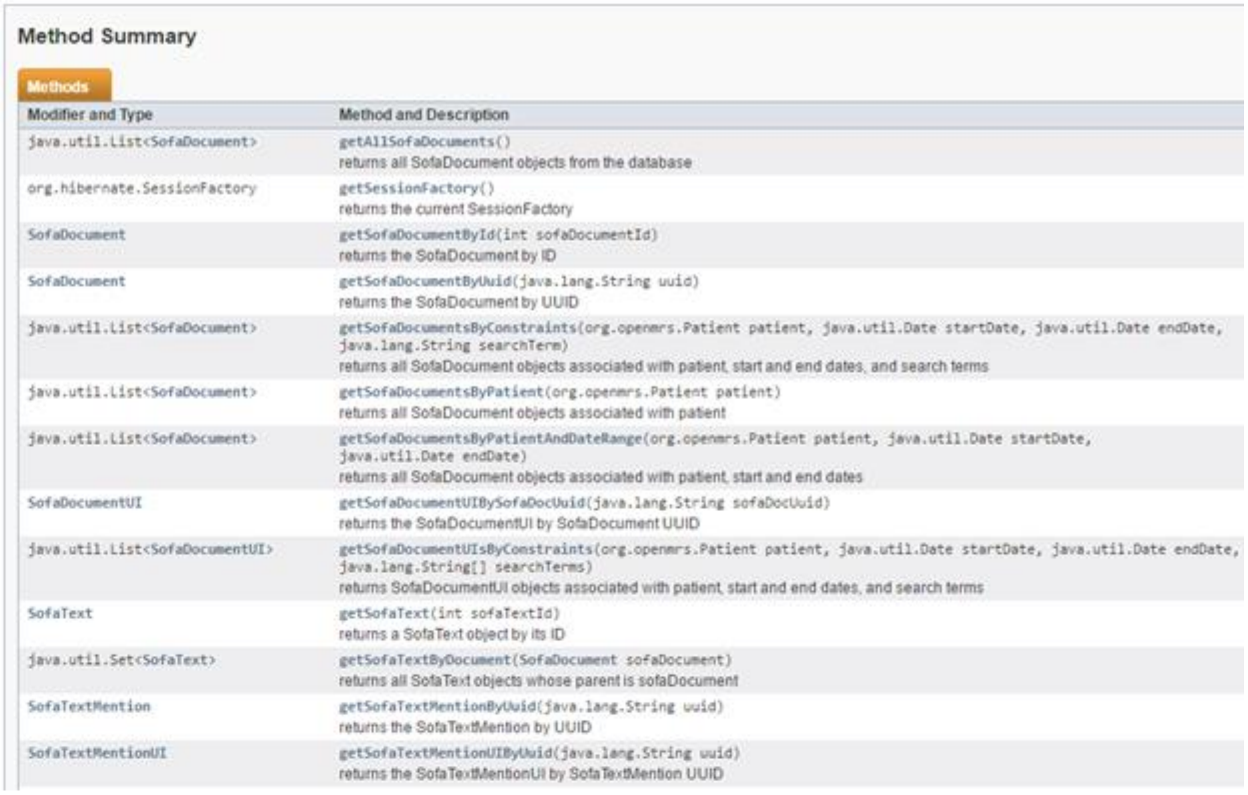
```
@Transactional
public interface NLPService
extends org.openmrs.api.OpenmrsService
```

This service exposes access to the data persistence layer for saving/retrieving SofaDocument related data. It is a Spring managed bean which is configured in moduleApplicationContext.xml.

It can be accessed only via Context:
`Context.getService(NLPService.class).someMethod();`

See Also:

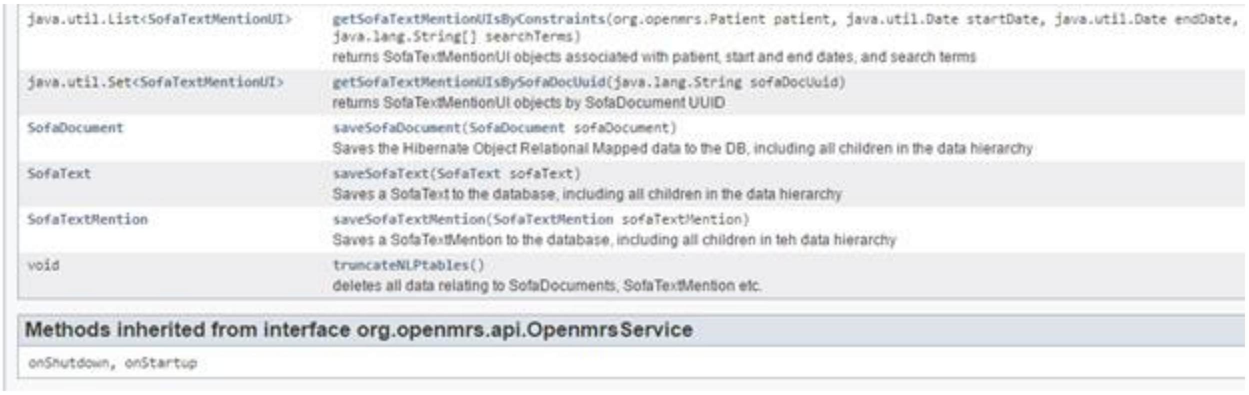
`Context`



Method Summary

Methods

Modifier and Type	Method and Description
`java.util.List<SofaDocument>`	`getAllSofaDocuments()` returns all SofaDocument objects from the database
`org.hibernate.SessionFactory`	`getSessionFactory()` returns the current SessionFactory
`SofaDocument`	`getSofaDocumentById(int sofaDocumentId)` returns the SofaDocument by ID
`SofaDocument`	`getSofaDocumentByUuid(java.lang.String uuid)` returns the SofaDocument by UUID
`java.util.List<SofaDocument>`	`getSofaDocumentsByConstraints(org.openmrs.Patient patient, java.util.Date startDate, java.util.Date endDate, java.lang.String searchTerm)` returns all SofaDocument objects associated with patient, start and end dates, and search terms
`java.util.List<SofaDocument>`	`getSofaDocumentsByPatient(org.openmrs.Patient patient)` returns all SofaDocument objects associated with patient
`java.util.List<SofaDocument>`	`getSofaDocumentsByPatientAndDateRange(org.openmrs.Patient patient, java.util.Date startDate, java.util.Date endDate)` returns all SofaDocument objects associated with patient, start and end dates
`SofaDocumentUI`	`getSofaDocumentUIBySofaDocUuid(java.lang.String sofaDocUuid)` returns the SofaDocumentUI by SofaDocument UUID
`java.util.List<SofaDocumentUI>`	`getSofaDocumentUIsByConstraints(org.openmrs.Patient patient, java.util.Date startDate, java.util.Date endDate, java.lang.String[] searchTerms)` returns SofaDocumentUI objects associated with patient, start and end dates, and search terms
`SofaText`	`getSofaText(int sofaTextId)` returns a SofaText object by its ID
`java.util.Set<SofaText>`	`getSofaTextByDocument(SofaDocument sofaDocument)` returns all SofaText objects whose parent is sofaDocument
`SofaTextMention`	`getSofaTextMentionByUuid(java.lang.String uuid)` returns the SofaTextMention by UUID
`SofaTextMentionUI`	`getSofaTextMentionUIByUuid(java.lang.String uuid)` returns the SofaTextMentionUI by SofaTextMention UUID



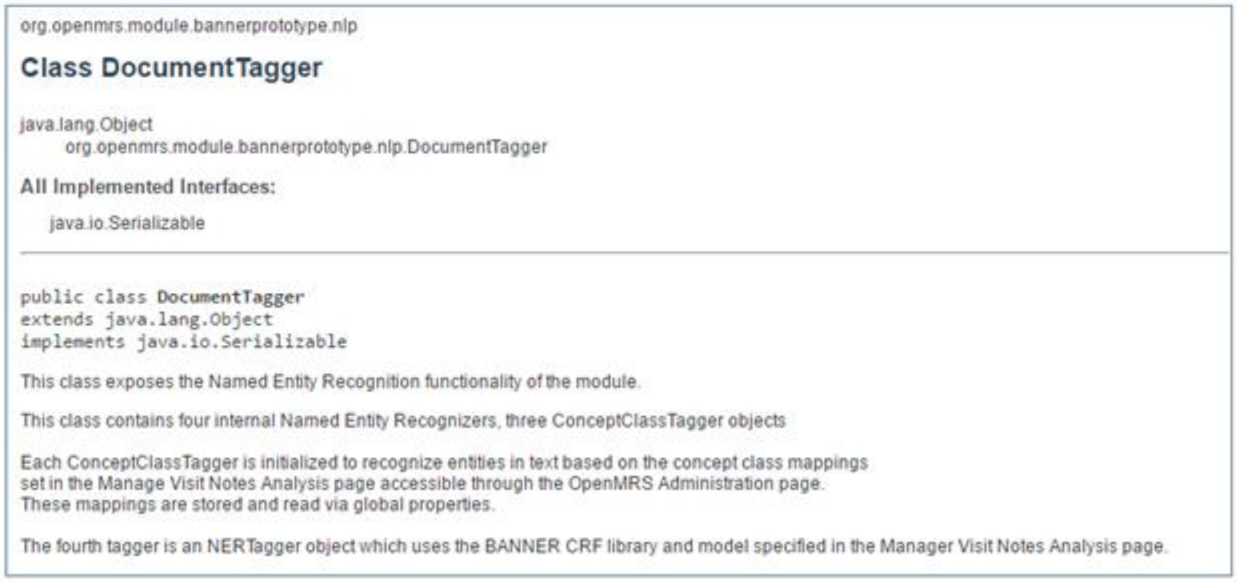
Modifier and Type	Method and Description
`java.util.List<SofaTextMentionUI>`	`getSofaTextMentionUIsByConstraints(org.openmrs.Patient patient, java.util.Date startDate, java.util.Date endDate, java.lang.String[] searchTerms)` returns SofaTextMentionUI objects associated with patient, start and end dates, and search terms
`java.util.Set<SofaTextMentionUI>`	`getSofaTextMentionUIsBySofaDocUuid(java.lang.String sofaDocUuid)` returns SofaTextMentionUI objects by SofaDocument UUID
`SofaDocument`	`saveSofaDocument(SofaDocument sofaDocument)` Saves the Hibernate Object Relational Mapped data to the DB, including all children in the data hierarchy
`SofaText`	`saveSofaText(SofaText sofaText)` Saves a SofaText to the database, including all children in the data hierarchy
`SofaTextMention`	`saveSofaTextMention(SofaTextMention sofaTextMention)` Saves a SofaTextMention to the database, including all children in teh data hierarchy
`void`	`truncateNLPtables()` deletes all data relating to SofaDocuments, SofaTextMention etc.

Methods inherited from interface org.openmrs.api.OpenmrsService

`onShutdown, onStartup`

Figure 10: Javadoc excerpts for NLPService

3.2.2 DocumentTagger


org.openmrs.module.bannerprototype.nlp

Class DocumentTagger

java.lang.Object
    org.openmrs.module.bannerprototype.nlp.DocumentTagger

All Implemented Interfaces:

java.io.Serializable

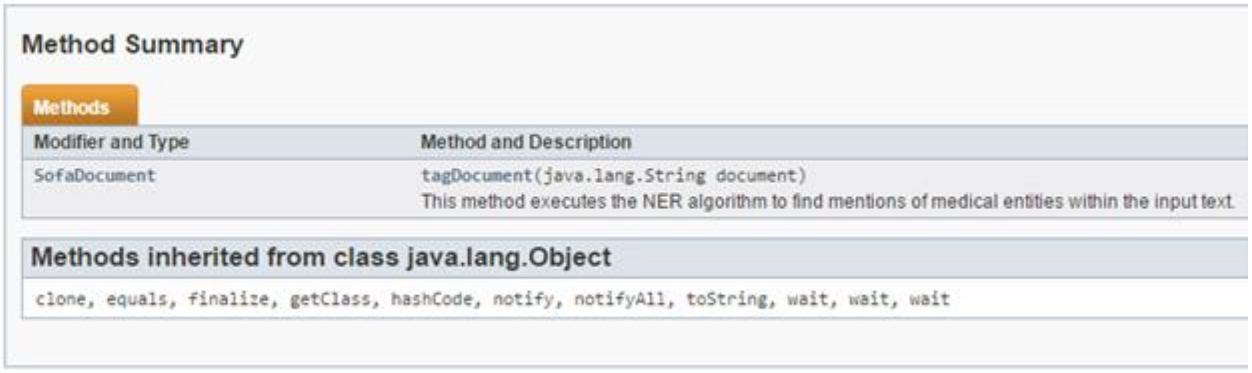
```
public class DocumentTagger
extends java.lang.Object
implements java.io.Serializable
```

This class exposes the Named Entity Recognition functionality of the module.

This class contains four internal Named Entity Recognizers, three ConceptClassTagger objects

Each ConceptClassTagger is initialized to recognize entities in text based on the concept class mappings set in the Manage Visit Notes Analysis page accessible through the OpenMRS Administration page. These mappings are stored and read via global properties.

The fourth tagger is an NERTagger object which uses the BANNER CRF library and model specified in the Manager Visit Notes Analysis page.

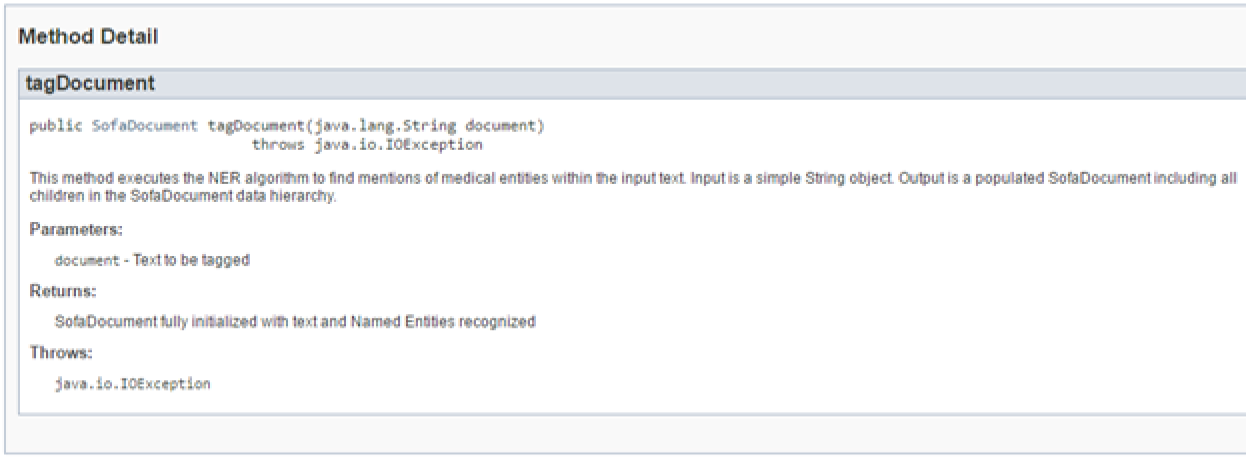

Method Summary

Methods

Modifier and Type	Method and Description
`SofaDocument`	`tagDocument(java.lang.String document)` This method executes the NER algorithm to find mentions of medical entities within the input text.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`


Method Detail

tagDocument

```
public SofaDocument tagDocument(java.lang.String document)
                         throws java.io.IOException
```

This method executes the NER algorithm to find mentions of medical entities within the input text. Input is a simple String object. Output is a populated SofaDocument including all children in the SofaDocument data hierarchy.

Parameters:

`document` - Text to be tagged

Returns:

SofaDocument fully initialized with text and Named Entities recognized

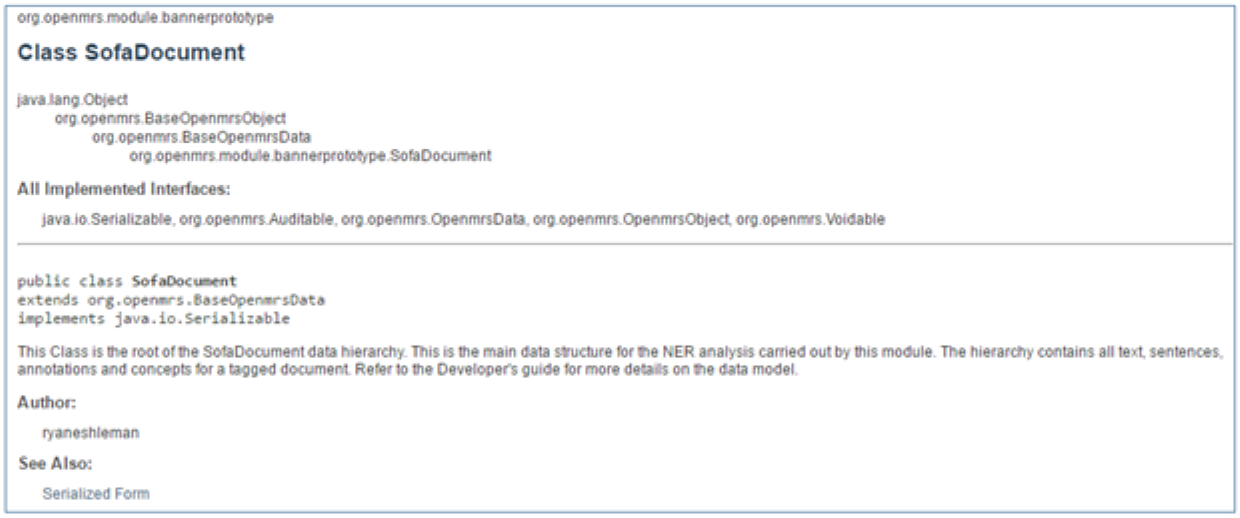
Throws:

`java.io.IOException`

Figure 11: Javadoc excerpts for DocumentTagger

3.2.3 SofaDocument

SofaDocument objects are created as a result of the DocumentTagger.tag() method. When a visit note is saved by OpenMRS an encounter is generated/saved along with it. Visit Notes are captured and analyzed via AOP when EncounterService.saveEncounter() is called. The encounter associated with this visit note is recorded in the SofaDocument.



org.openmrs.module.bannerprototype

Class SofaDocument

```
java.lang.Object
    org.openmrs.BaseOpenmrsObject
        org.openmrs.BaseOpenmrsData
            org.openmrs.module.bannerprototype.SofaDocument
```

All Implemented Interfaces:

java.io.Serializable, org.openmrs.Auditable, org.openmrs.OpenmrsData, org.openmrs.OpenmrsObject, org.openmrs.Voidable

```
public class SofaDocument
extends org.openmrs.BaseOpenmrsData
implements java.io.Serializable
```

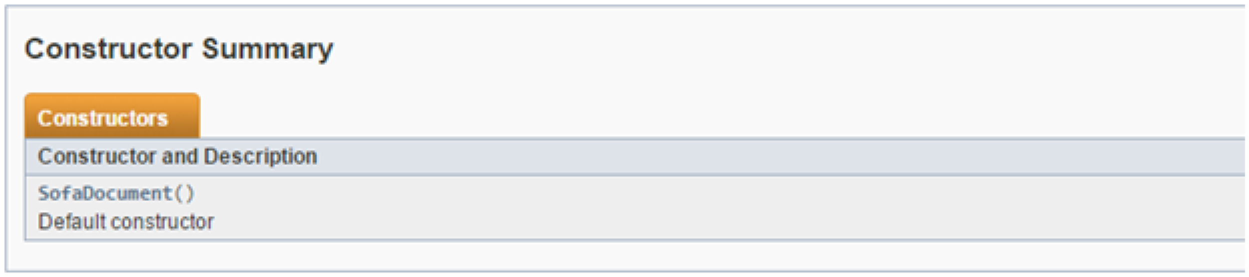
This Class is the root of the SofaDocument data hierarchy. This is the main data structure for the NER analysis carried out by this module. The hierarchy contains all text, sentences, annotations and concepts for a tagged document. Refer to the Developer's guide for more details on the data model.

Author:

ryaneshleman

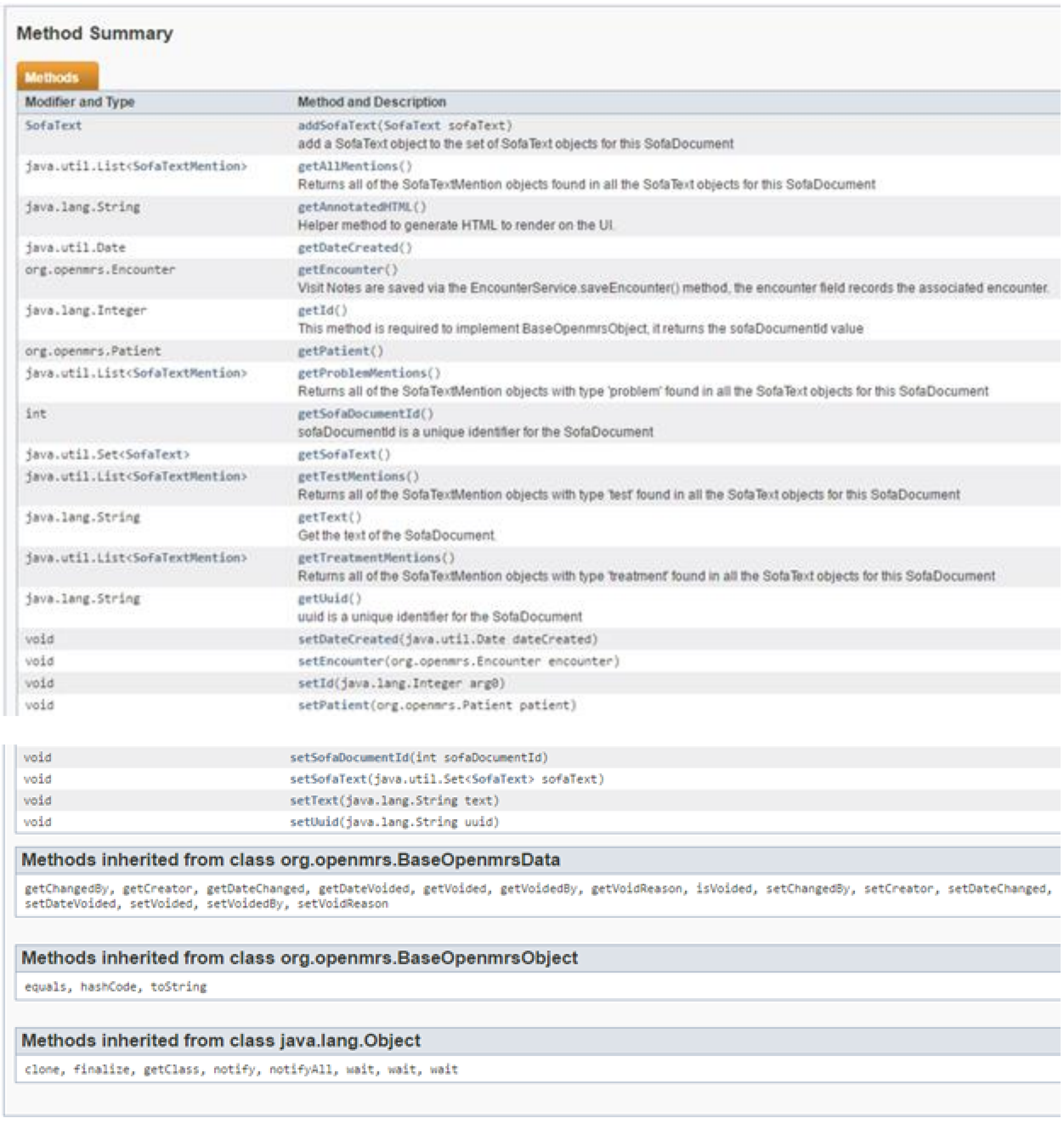
See Also:

Serialized Form



Constructor Summary

Constructor and Description
`SofaDocument()` Default constructor



Method Summary

Methods

Modifier and Type	Method and Description
SofaText	addSofaText(SofaText sofaText) add a SofaText object to the set of SofaText objects for this SofaDocument
java.util.List<SofaTextMention>	getAllMentions() Returns all of the SofaTextMention objects found in all the SofaText objects for this SofaDocument
java.lang.String	getAnnotatedHTML() Helper method to generate HTML to render on the UI.
java.util.Date	getDateCreated()
org.openmrs.Encounter	getEncounter() Visit Notes are saved via the EncounterService.saveEncounter() method, the encounter field records the associated encounter.
java.lang.Integer	getId() This method is required to implement BaseOpenmrsObject, it returns the sofaDocumentId value
org.openmrs.Patient	getPatient()
java.util.List<SofaTextMention>	getProblemMentions() Returns all of the SofaTextMention objects with type 'problem' found in all the SofaText objects for this SofaDocument
int	getSofaDocumentId() sofaDocumentId is a unique identifier for the SofaDocument
java.util.Set<SofaText>	getSofaText()
java.util.List<SofaTextMention>	getTestMentions() Returns all of the SofaTextMention objects with type 'test' found in all the SofaText objects for this SofaDocument
java.lang.String	getText() Get the text of the SofaDocument.
java.util.List<SofaTextMention>	getTreatmentMentions() Returns all of the SofaTextMention objects with type 'treatment' found in all the SofaText objects for this SofaDocument
java.lang.String	getUuid() uuid is a unique identifier for the SofaDocument
void	setDateCreated(java.util.Date dateCreated)
void	setEncounter(org.openmrs.Encounter encounter)
void	setId(java.lang.Integer arg0)
void	setPatient(org.openmrs.Patient patient)
void	setSofaDocumentId(int sofaDocumentId)
void	setSofaText(java.util.Set<SofaText> sofaText)
void	setText(java.lang.String text)
void	setUuid(java.lang.String uuid)

Methods inherited from class org.openmrs.BaseOpenmrsData

getChangedBy, getCreator, getDateChanged, getDateVoided, getVoided, getVoidedBy, getVoidReason, isVoided, setChangedBy, setCreator, setDateChanged, setDateVoided, setVoided, setVoidedBy, setVoidReason

Methods inherited from class org.openmrs.BaseOpenmrsObject

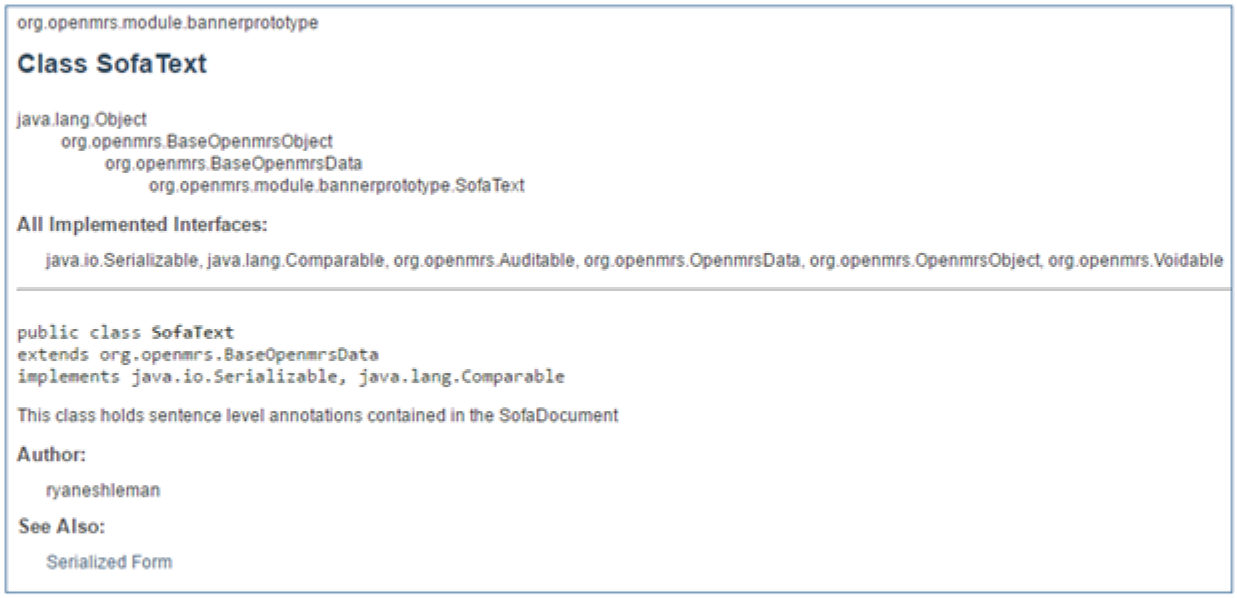
equals, hashCode, toString

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Figure 12: Javadoc excerpts for SofaDocument class

3.2.4 SofaText


org.openmrs.module.bannerprototype

Class SofaText

```
java.lang.Object
    org.openmrs.BaseOpenmrsObject
        org.openmrs.BaseOpenmrsData
            org.openmrs.module.bannerprototype.SofaText
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable, org.openmrs.Auditable, org.openmrs.OpenmrsData, org.openmrs.OpenmrsObject, org.openmrs.Voidable

```
public class SofaText
extends org.openmrs.BaseOpenmrsData
implements java.io.Serializable, java.lang.Comparable
```

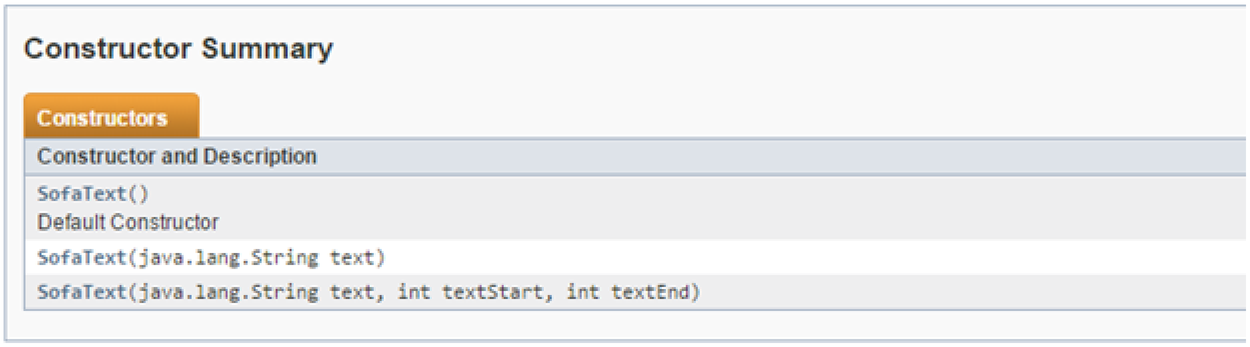
This class holds sentence level annotations contained in the SofaDocument

Author:

ryaneshleman

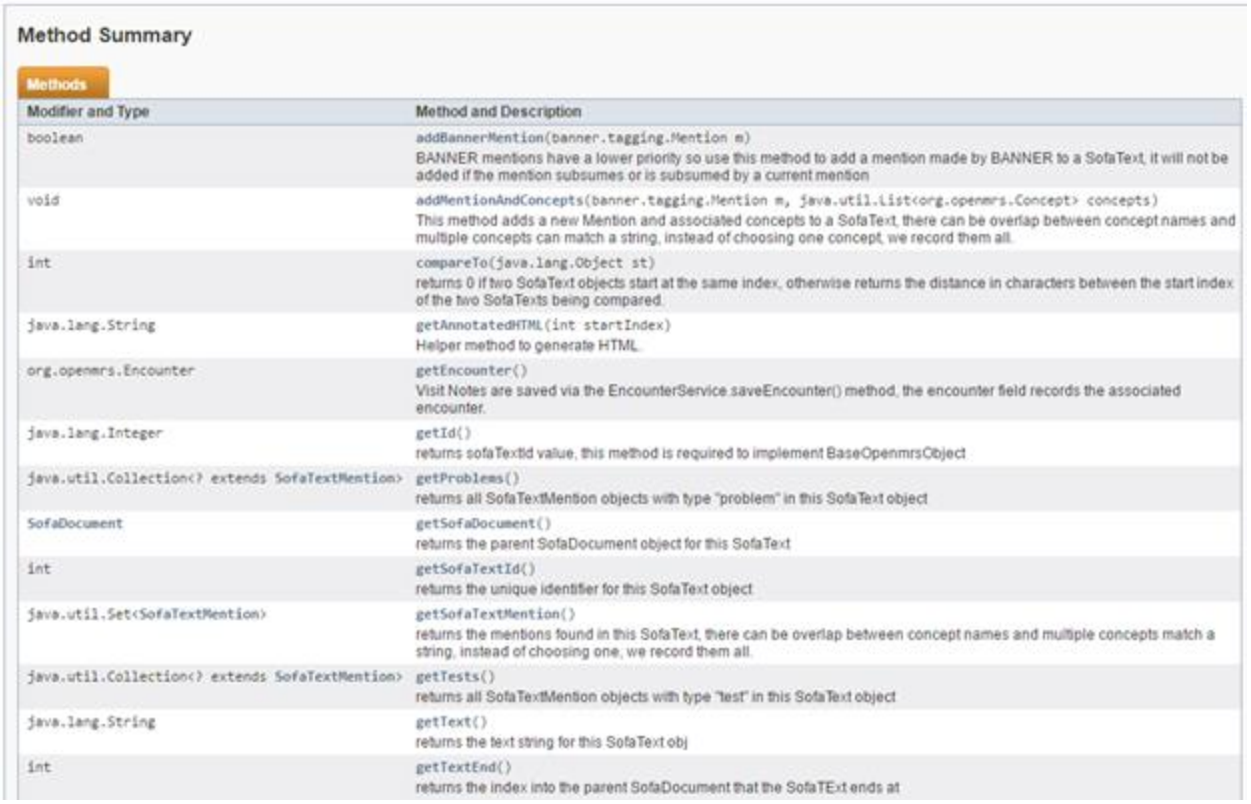
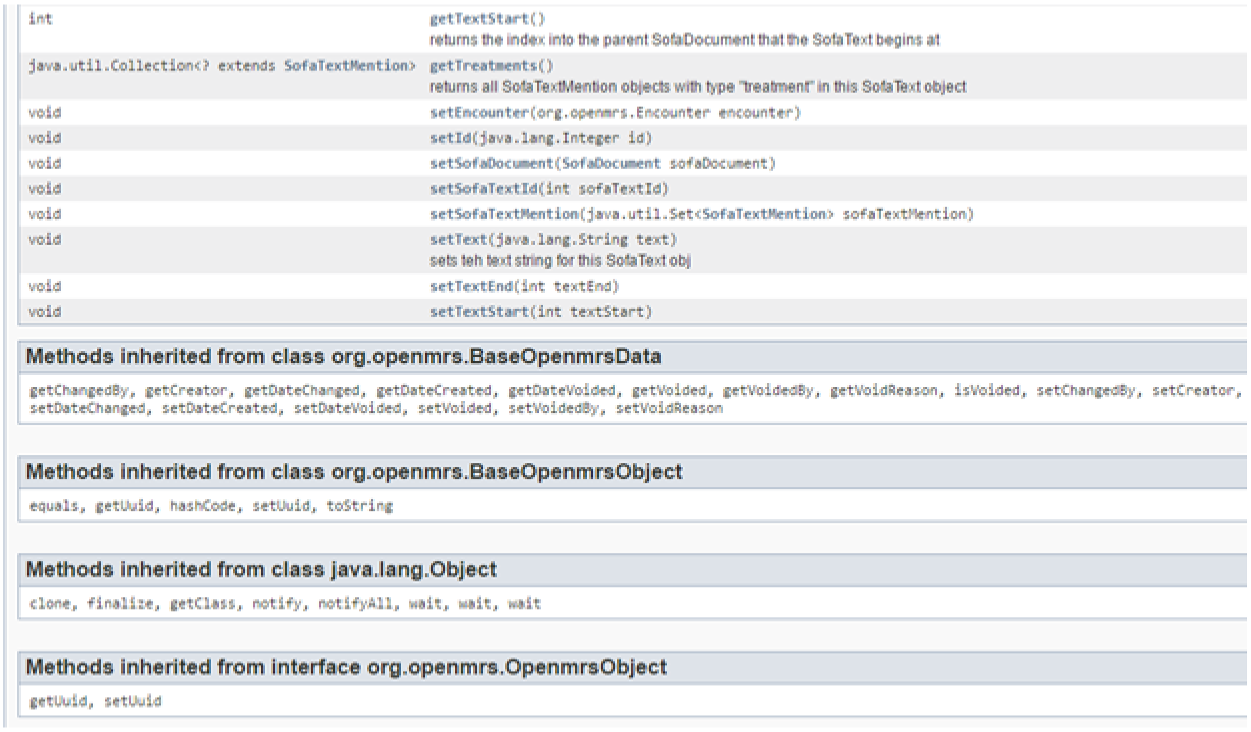
See Also:

Serialized Form


Constructor Summary

Constructors

Constructor and Description
`SofaText()` Default Constructor
`SofaText(java.lang.String text)`
`SofaText(java.lang.String text, int textStart, int textEnd)`

Method Summary

Methods

Modifier and Type	Method and Description
boolean	addBannerMention(banner.tagging.Mention m) BANNER mentions have a lower priority so use this method to add a mention made by BANNER to a SofaText, it will not be added if the mention subsumes or is subsumed by a current mention
void	addMentionAndConcepts(banner.tagging.Mention m, java.util.List<org.openmrs.Concept> concepts) This method adds a new Mention and associated concepts to a SofaText, there can be overlap between concept names and multiple concepts can match a string, instead of choosing one concept, we record them all.
int	compareTo(java.lang.Object st) returns 0 if two SofaText objects start at the same index, otherwise returns the distance in characters between the start index of the two SofaTexts being compared.
java.lang.String	getAnnotatedHTML(int startIndex) Helper method to generate HTML.
org.openmrs.Encounter	getEncounter() Visit Notes are saved via the EncounterService.saveEncounter() method, the encounter field records the associated encounter.
java.lang.Integer	getId() returns sofaTextId value, this method is required to implement BaseOpenmrsObject
java.util.Collection<? extends SofaTextMention>	getProblems() returns all SofaTextMention objects with type "problem" in this SofaText object
SofaDocument	getSofaDocument() returns the parent SofaDocument object for this SofaText
int	getSofaTextId() returns the unique identifier for this SofaText object
java.util.Set<SofaTextMention>	getSofaTextMention() returns the mentions found in this SofaText, there can be overlap between concept names and multiple concepts match a string, instead of choosing one, we record them all.
java.util.Collection<? extends SofaTextMention>	getTests() returns all SofaTextMention objects with type "test" in this SofaText object
java.lang.String	getText() returns the text string for this SofaText obj
int	getTextEnd() returns the index into the parent SofaDocument that the SofaTExt ends at
int	getTextStart() returns the index into the parent SofaDocument that the SofaText begins at
java.util.Collection<? extends SofaTextMention>	getTreatments() returns all SofaTextMention objects with type "treatment" in this SofaText object
void	setEncounter(org.openmrs.Encounter encounter)
void	setId(java.lang.Integer id)
void	setSofaDocument(SofaDocument sofaDocument)
void	setSofaTextId(int sofaTextId)
void	setSofaTextMention(java.util.Set<SofaTextMention> sofaTextMention)
void	setText(java.lang.String text) sets teh text string for this SofaText obj
void	setTextEnd(int textEnd)
void	setTextStart(int textStart)

Methods inherited from class org.openmrs.BaseOpenmrsData

getChangedBy, getCreator, getDateChanged, getDateCreated, getDateVoided, getVoided, getVoidedBy, getVoidReason, isVoided, setChangedBy, setCreator, setDateChanged, setDateCreated, setDateVoided, setVoided, setVoidedBy, setVoidReason

Methods inherited from class org.openmrs.BaseOpenmrsObject

equals, getUuid, hashCode, setUuid, toString

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

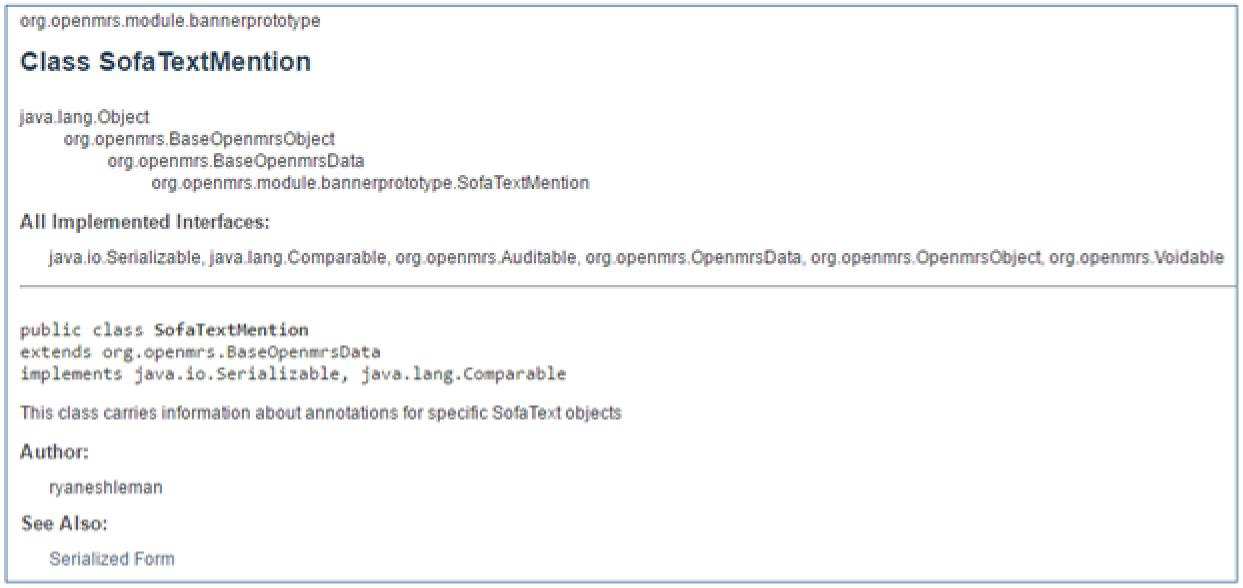
Methods inherited from interface org.openmrs.OpenmrsObject

getUuid, setUuid

Figure 13: Javadoc excerpts for SofaText class

3.2.5 SofaTextMention

The NER algorithm executed in DocumentTagger.tag() may find multiple OpenMRS Concepts in a single mention. This is a consequence of overlapping concept names/synonyms in the Concept Dictionary. Instead of forcing the algorithm to choose one Concept, we record them all.



org.openmrs.module.bannerprototype

Class SofaTextMention

```
java.lang.Object
    org.openmrs.BaseOpenmrsObject
        org.openmrs.BaseOpenmrsData
            org.openmrs.module.bannerprototype.SofaTextMention
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable, org.openmrs.Auditable, org.openmrs.OpenmrsData, org.openmrs.OpenmrsObject, org.openmrs.Voidable

```
public class SofaTextMention
extends org.openmrs.BaseOpenmrsData
implements java.io.Serializable, java.lang.Comparable
```

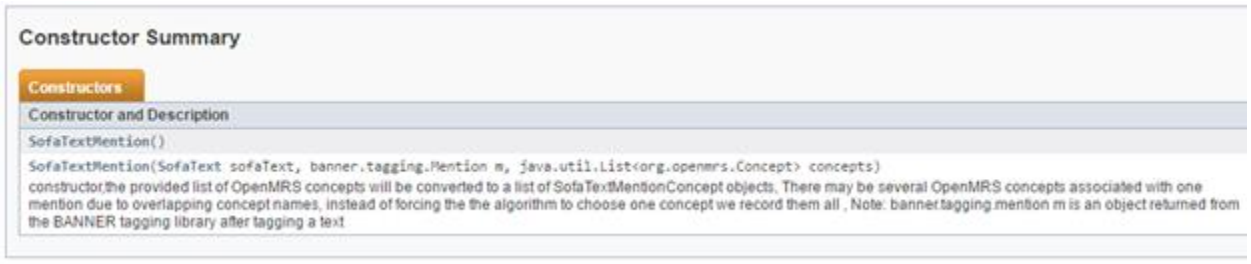
This class carries information about annotations for specific SofaText objects

Author:

ryaneshleman

See Also:

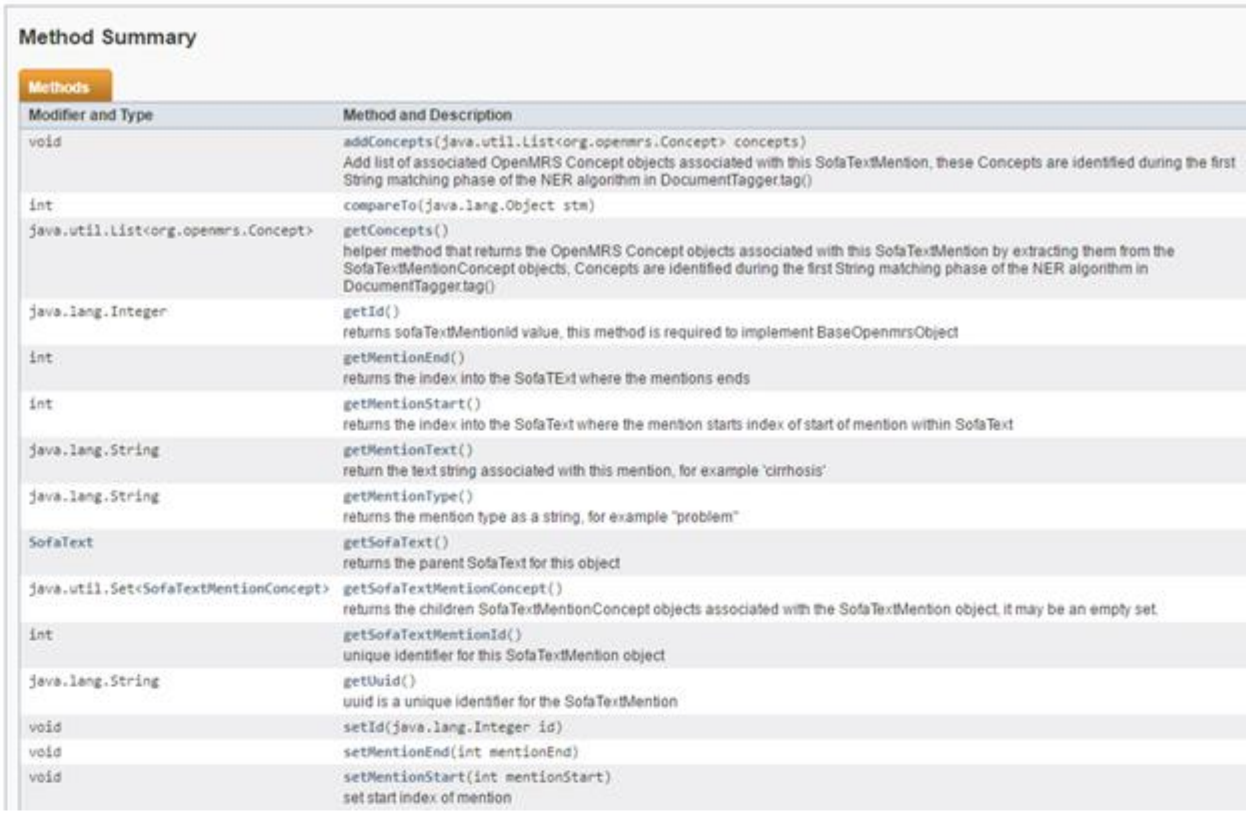
Serialized Form



Constructor Summary

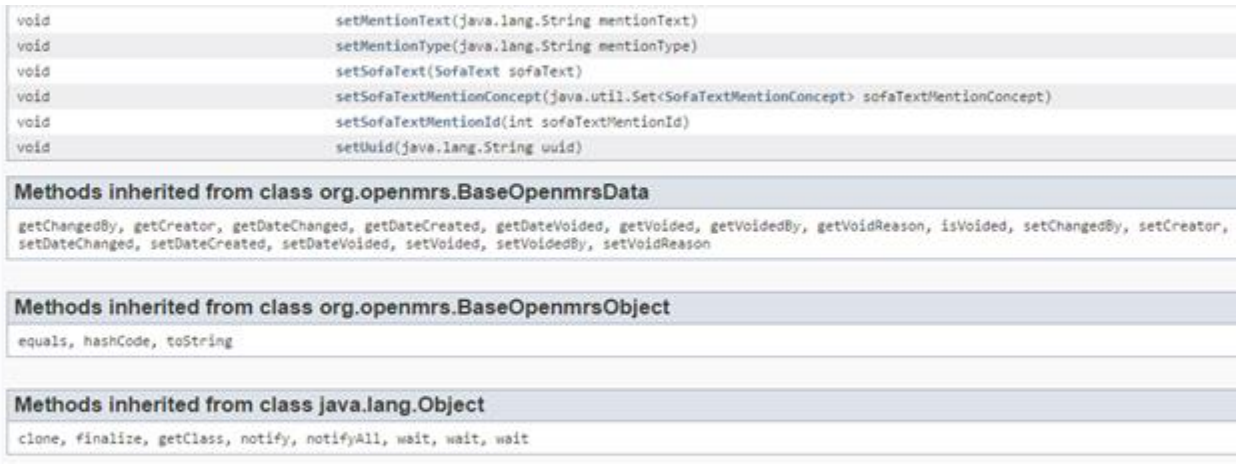
Constructors

Constructor and Description
`SofaTextMention()`
`SofaTextMention(SofaText sofaText, banner.tagging.Mention m, java.util.List<org.openmrs.Concept> concepts)` constructor,the provided list of OpenMRS concepts will be converted to a list of SofaTextMentionConcept objects. There may be several OpenMRS concepts associated with one mention due to overlapping concept names, instead of forcing the the algorithm to choose one concept we record them all , Note: banner.tagging.mention m is an object returned from the BANNER tagging library after tagging a text


Method Summary

Methods

Modifier and Type	Method and Description
`void`	`addConcepts(java.util.List<org.openmrs.Concept> concepts)` Add list of associated OpenMRS Concept objects associated with this SofaTextMention, these Concepts are identified during the first String matching phase of the NER algorithm in DocumentTagger.tag()
`int`	`compareTo(java.lang.Object stm)`
`java.util.List<org.openmrs.Concept>`	`getConcepts()` helper method that returns the OpenMRS Concept objects associated with this SofaTextMention by extracting them from the SofaTextMentionConcept objects. Concepts are identified during the first String matching phase of the NER algorithm in DocumentTagger.tag()
`java.lang.Integer`	`getId()` returns sofaTextMentionId value, this method is required to implement BaseOpenmrsObject
`int`	`getMentionEnd()` returns the index into the SofaTExt where the mentions ends
`int`	`getMentionStart()` returns the index into the SofaText where the mention starts index of start of mention within SofaText
`java.lang.String`	`getMentionText()` return the text string associated with this mention, for example 'cirrhosis'
`java.lang.String`	`getMentionType()` returns the mention type as a string, for example "problem"
`SofaText`	`getSofaText()` returns the parent SofaText for this object
`java.util.Set<SofaTextMentionConcept>`	`getSofaTextMentionConcept()` returns the children SofaTextMentionConcept objects associated with the SofaTextMention object, it may be an empty set.
`int`	`getSofaTextMentionId()` unique identifier for this SofaTextMention object
`java.lang.String`	`getUuid()` uuid is a unique identifier for the SofaTextMention
`void`	`setId(java.lang.Integer id)`
`void`	`setMentionEnd(int mentionEnd)`
`void`	`setMentionStart(int mentionStart)` set start index of mention



`void`	`setMentionText(java.lang.String mentionText)`
`void`	`setMentionType(java.lang.String mentionType)`
`void`	`setSofaText(SofaText sofaText)`
`void`	`setSofaTextMentionConcept(java.util.Set<SofaTextMentionConcept> sofaTextMentionConcept)`
`void`	`setSofaTextMentionId(int sofaTextMentionId)`
`void`	`setUuid(java.lang.String uuid)`

Methods inherited from class org.openmrs.BaseOpenmrsData

```
getChangedBy, getCreator, getDateChanged, getDateCreated, getDateVoided, getVoided, getVoidedBy, getVoidReason, isVoided, setChangedBy, setCreator,
setDateChanged, setDateCreated, setDateVoided, setVoided, setVoidedBy, setVoidReason
```

Methods inherited from class org.openmrs.BaseOpenmrsObject

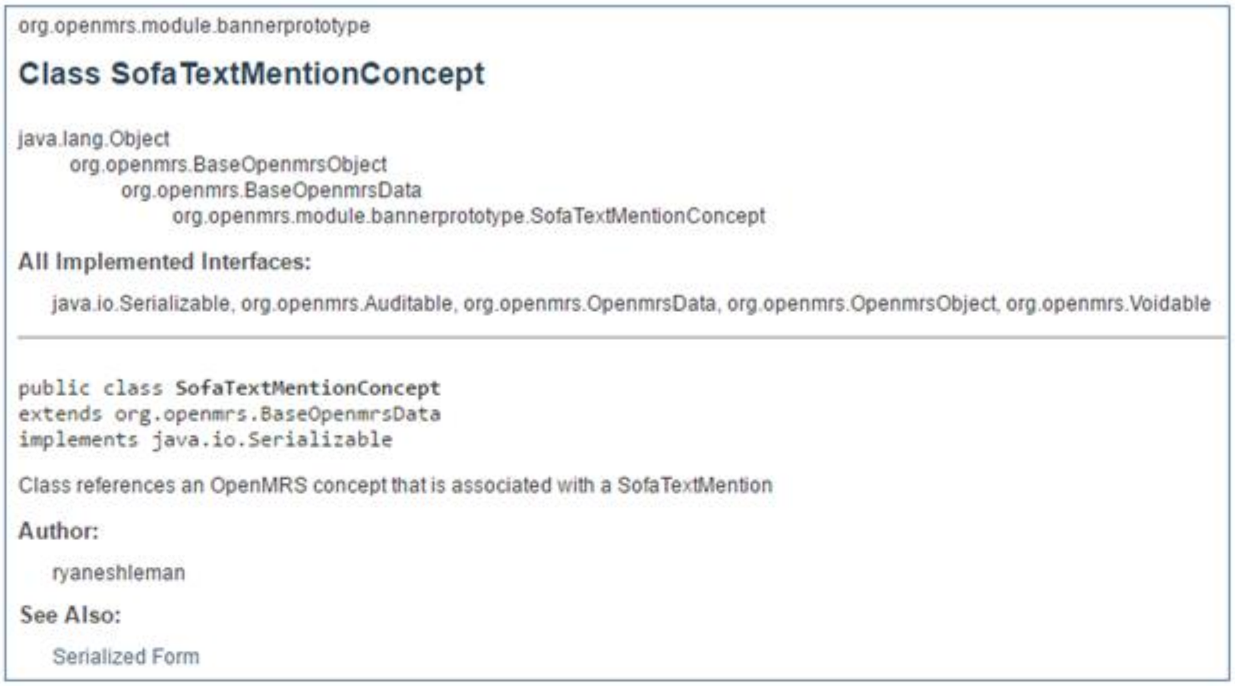
```
equals, hashCode, toString
```

Methods inherited from class java.lang.Object

```
clone, finalize, getClass, notify, notifyAll, wait, wait, wait
```

Figure 14: Javadoc excerpts for SofaTextMention class

3.2.6 SofaTextMentionConcept



org.openmrs.module.bannerprototype

Class SofaTextMentionConcept

```
java.lang.Object
    org.openmrs.BaseOpenmrsObject
        org.openmrs.BaseOpenmrsData
            org.openmrs.module.bannerprototype.SofaTextMentionConcept
```

All Implemented Interfaces:

java.io.Serializable, org.openmrs.Auditable, org.openmrs.OpenmrsData, org.openmrs.OpenmrsObject, org.openmrs.Voidable

```
public class SofaTextMentionConcept
extends org.openmrs.BaseOpenmrsData
implements java.io.Serializable
```

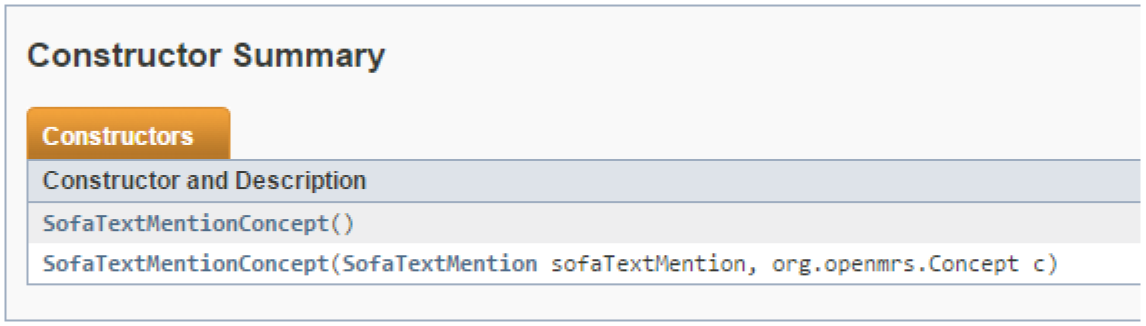
Class references an OpenMRS concept that is associated with a SofaTextMention

Author:

ryaneshleman

See Also:

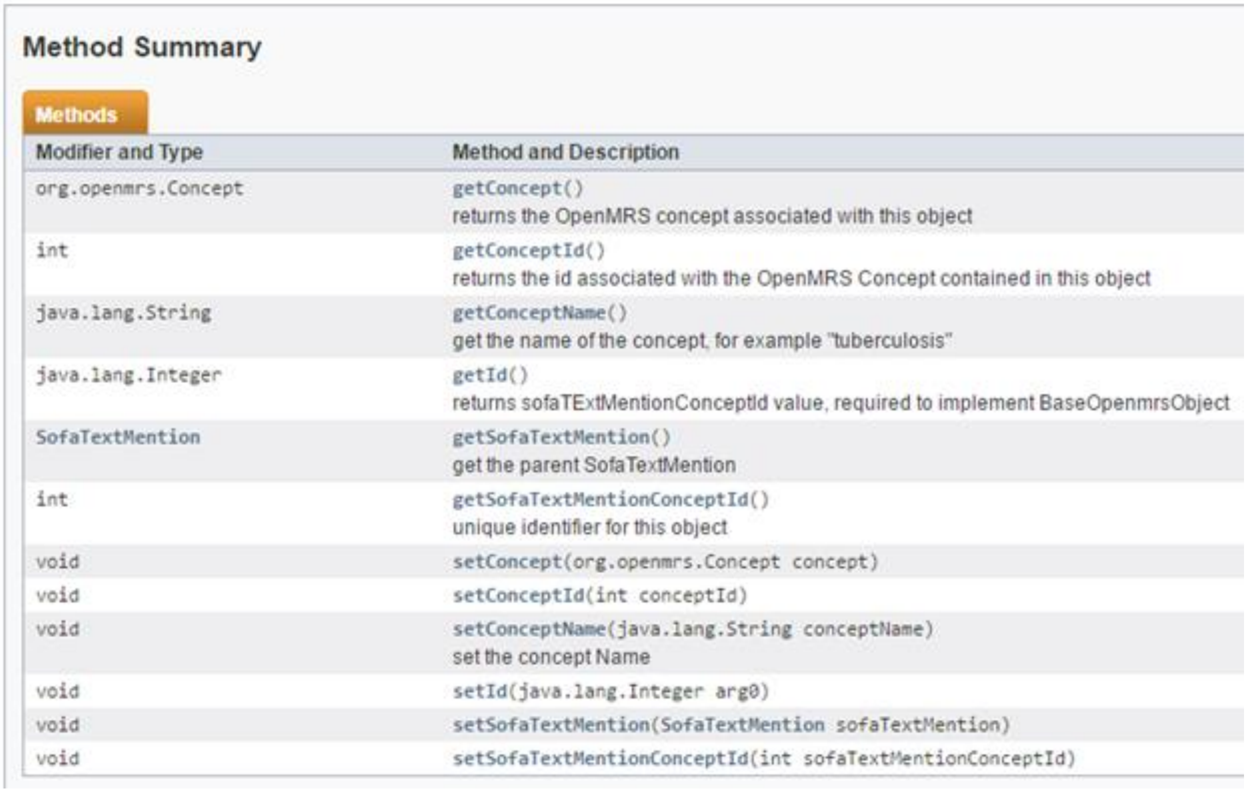
Serialized Form



Constructor Summary

Constructors

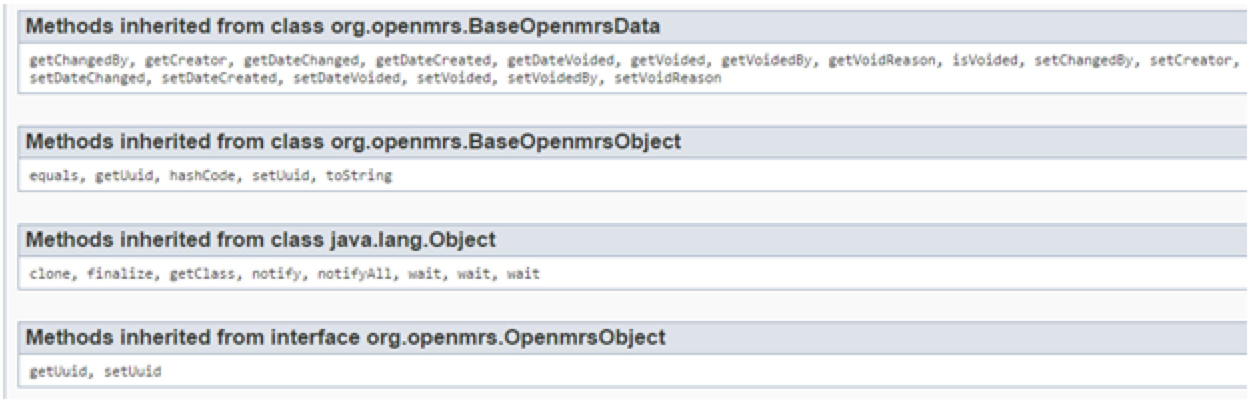
Constructor and Description
`SofaTextMentionConcept()`
`SofaTextMentionConcept(SofaTextMention sofaTextMention, org.openmrs.Concept c)`



Method Summary

Methods

Modifier and Type	Method and Description
`org.openmrs.Concept`	`getConcept()` returns the OpenMRS concept associated with this object
`int`	`getConceptId()` returns the id associated with the OpenMRS Concept contained in this object
`java.lang.String`	`getConceptName()` get the name of the concept, for example "tuberculosis"
`java.lang.Integer`	`getId()` returns sofaTExtMentionConceptId value, required to implement BaseOpenmrsObject
`SofaTextMention`	`getSofaTextMention()` get the parent SofaTextMention
`int`	`getSofaTextMentionConceptId()` unique identifier for this object
`void`	`setConcept(org.openmrs.Concept concept)`
`void`	`setConceptId(int conceptId)`
`void`	`setConceptName(java.lang.String conceptName)` set the concept Name
`void`	`setId(java.lang.Integer arg0)`
`void`	`setSofaTextMention(SofaTextMention sofaTextMention)`
`void`	`setSofaTextMentionConceptId(int sofaTextMentionConceptId)`



Methods inherited from class org.openmrs.BaseOpenmrsData

`getChangedBy, getCreator, getDateChanged, getDateCreated, getDateVoided, getVoided, getVoidedBy, getVoidReason, isVoided, setChangedBy, setCreator, setDateChanged, setDateCreated, setDateVoided, setVoided, setVoidedBy, setVoidReason`

Methods inherited from class org.openmrs.BaseOpenmrsObject

`equals, getUuid, hashCode, setUuid, toString`

Methods inherited from class java.lang.Object

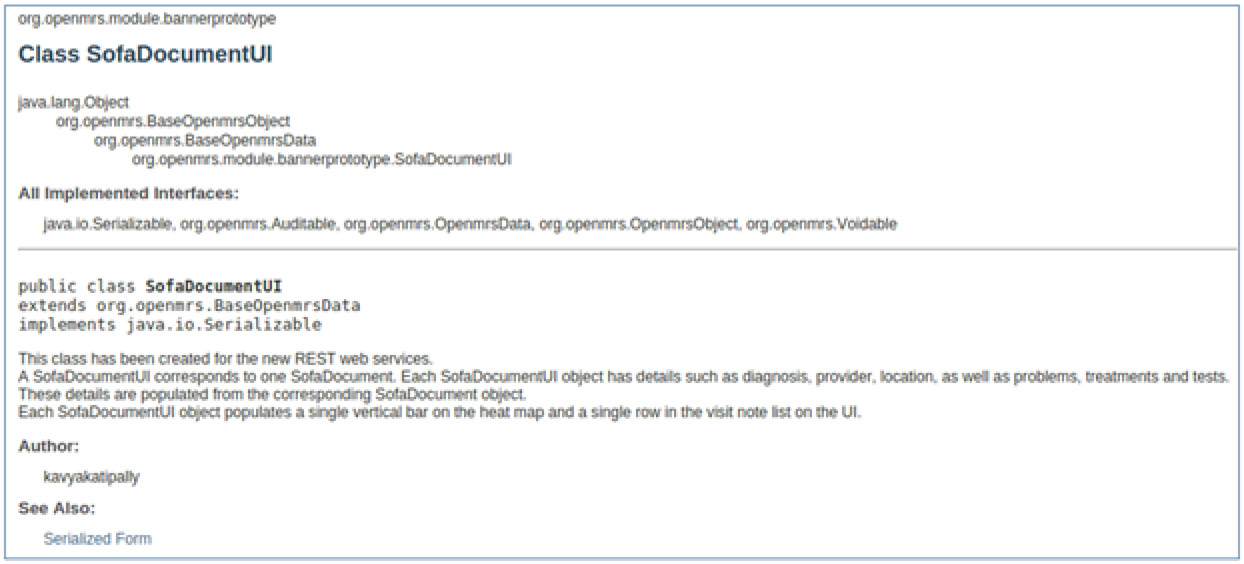
`clone, finalize, getClass, notify, notifyAll, wait, wait, wait`

Methods inherited from interface org.openmrs.OpenmrsObject

`getUuid, setUuid`

Figure 15: Javadoc excerpts for SofaTextMentionConcept class

3.2.7 SofaDocumentUI



org.openmrs.module.bannerprototype

Class SofaDocumentUI

```
java.lang.Object
    org.openmrs.BaseOpenmrsObject
        org.openmrs.BaseOpenmrsData
            org.openmrs.module.bannerprototype.SofaDocumentUI
```

All Implemented Interfaces:

java.io.Serializable, org.openmrs.Auditable, org.openmrs.OpenmrsData, org.openmrs.OpenmrsObject, org.openmrs.Voidable

```
public class SofaDocumentUI
extends org.openmrs.BaseOpenmrsData
implements java.io.Serializable
```

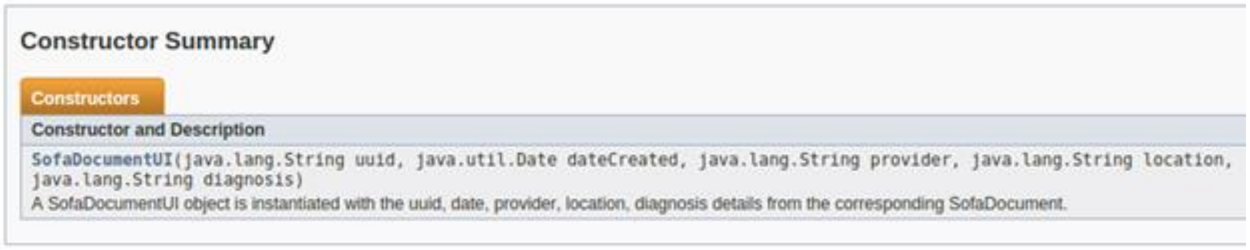
This class has been created for the new REST web services.
A SofaDocumentUI corresponds to one SofaDocument. Each SofaDocumentUI object has details such as diagnosis, provider, location, as well as problems, treatments and tests. These details are populated from the corresponding SofaDocument object.
Each SofaDocumentUI object populates a single vertical bar on the heat map and a single row in the visit note list on the UI.

Author:

kavyakatipally

See Also:

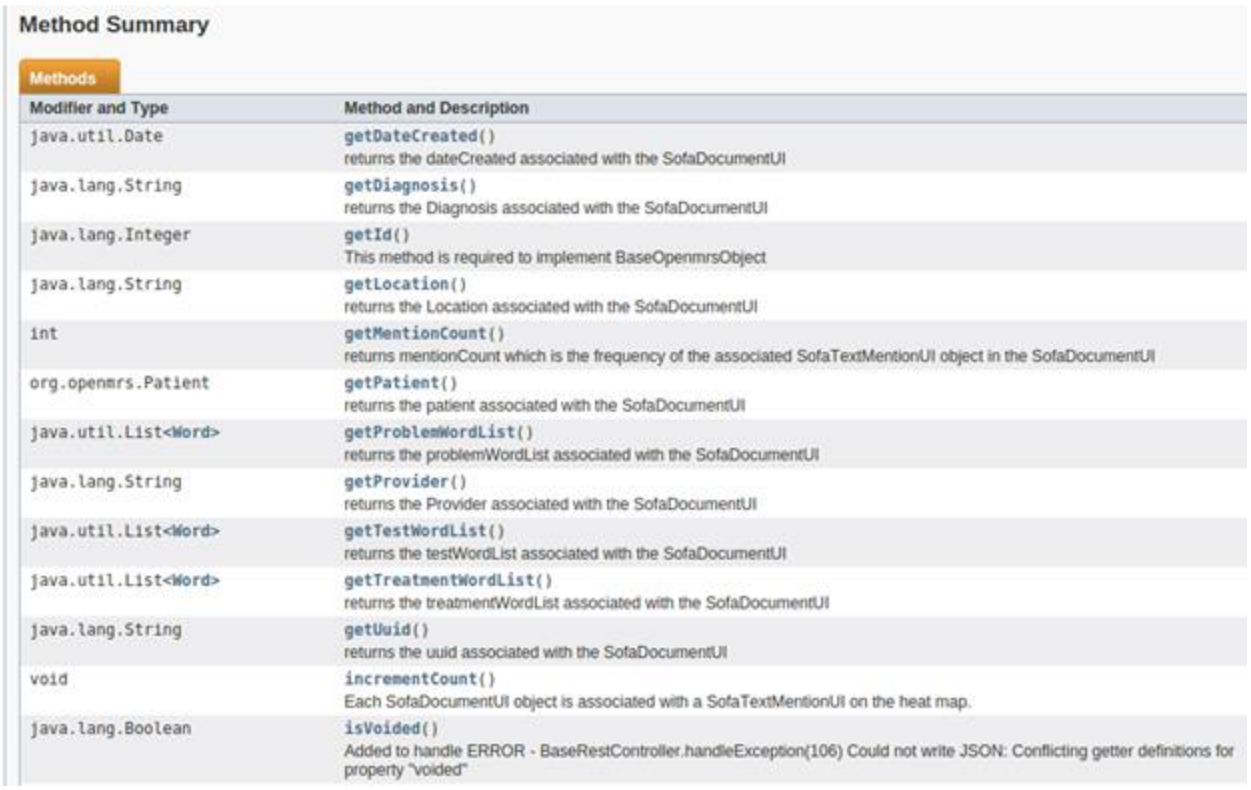
Serialized Form



Constructor Summary

Constructors

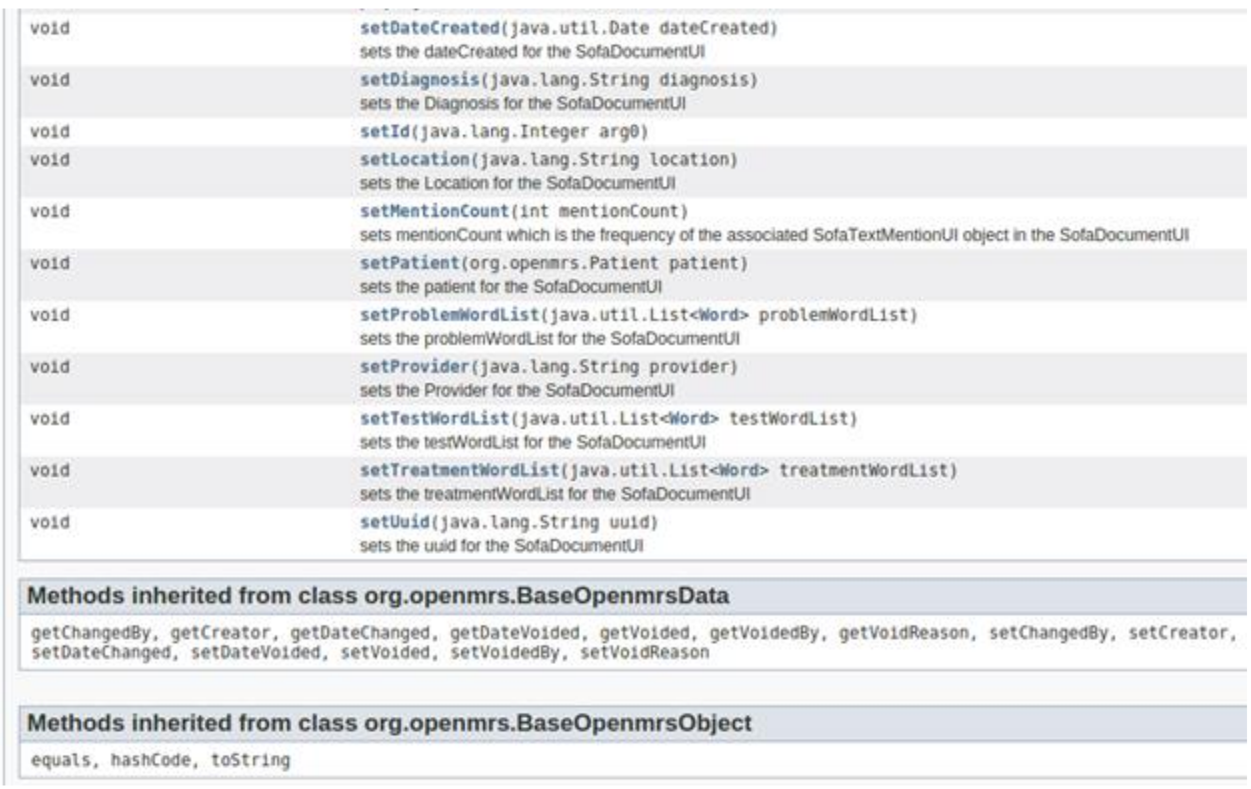
Constructor and Description
`SofaDocumentUI(java.lang.String uuid, java.util.Date dateCreated, java.lang.String provider, java.lang.String location, java.lang.String diagnosis)` A SofaDocumentUI object is instantiated with the uuid, date, provider, location, diagnosis details from the corresponding SofaDocument.



Method Summary

Methods

Modifier and Type	Method and Description
`java.util.Date`	`getDateCreated()` returns the dateCreated associated with the SofaDocumentUI
`java.lang.String`	`getDiagnosis()` returns the Diagnosis associated with the SofaDocumentUI
`java.lang.Integer`	`getId()` This method is required to implement BaseOpenmrsObject
`java.lang.String`	`getLocation()` returns the Location associated with the SofaDocumentUI
`int`	`getMentionCount()` returns mentionCount which is the frequency of the associated SofaTextMentionUI object in the SofaDocumentUI
`org.openmrs.Patient`	`getPatient()` returns the patient associated with the SofaDocumentUI
`java.util.List<Word>`	`getProblemWordList()` returns the problemWordList associated with the SofaDocumentUI
`java.lang.String`	`getProvider()` returns the Provider associated with the SofaDocumentUI
`java.util.List<Word>`	`getTestWordList()` returns the testWordList associated with the SofaDocumentUI
`java.util.List<Word>`	`getTreatmentWordList()` returns the treatmentWordList associated with the SofaDocumentUI
`java.lang.String`	`getUuid()` returns the uuid associated with the SofaDocumentUI
`void`	`incrementCount()` Each SofaDocumentUI object is associated with a SofaTextMentionUI on the heat map.
`java.lang.Boolean`	`isVoided()` Added to handle ERROR - BaseRestController.handleException(106) Could not write JSON: Conflicting getter definitions for property "voided"



`void`	`setDateCreated(java.util.Date dateCreated)` sets the dateCreated for the SofaDocumentUI
`void`	`setDiagnosis(java.lang.String diagnosis)` sets the Diagnosis for the SofaDocumentUI
`void`	`setId(java.lang.Integer arg0)`
`void`	`setLocation(java.lang.String location)` sets the Location for the SofaDocumentUI
`void`	`setMentionCount(int mentionCount)` sets mentionCount which is the frequency of the associated SofaTextMentionUI object in the SofaDocumentUI
`void`	`setPatient(org.openmrs.Patient patient)` sets the patient for the SofaDocumentUI
`void`	`setProblemWordList(java.util.List<Word> problemWordList)` sets the problemWordList for the SofaDocumentUI
`void`	`setProvider(java.lang.String provider)` sets the Provider for the SofaDocumentUI
`void`	`setTestWordList(java.util.List<Word> testWordList)` sets the testWordList for the SofaDocumentUI
`void`	`setTreatmentWordList(java.util.List<Word> treatmentWordList)` sets the treatmentWordList for the SofaDocumentUI
`void`	`setUuid(java.lang.String uuid)` sets the uuid for the SofaDocumentUI

Methods inherited from class org.openmrs.BaseOpenmrsData

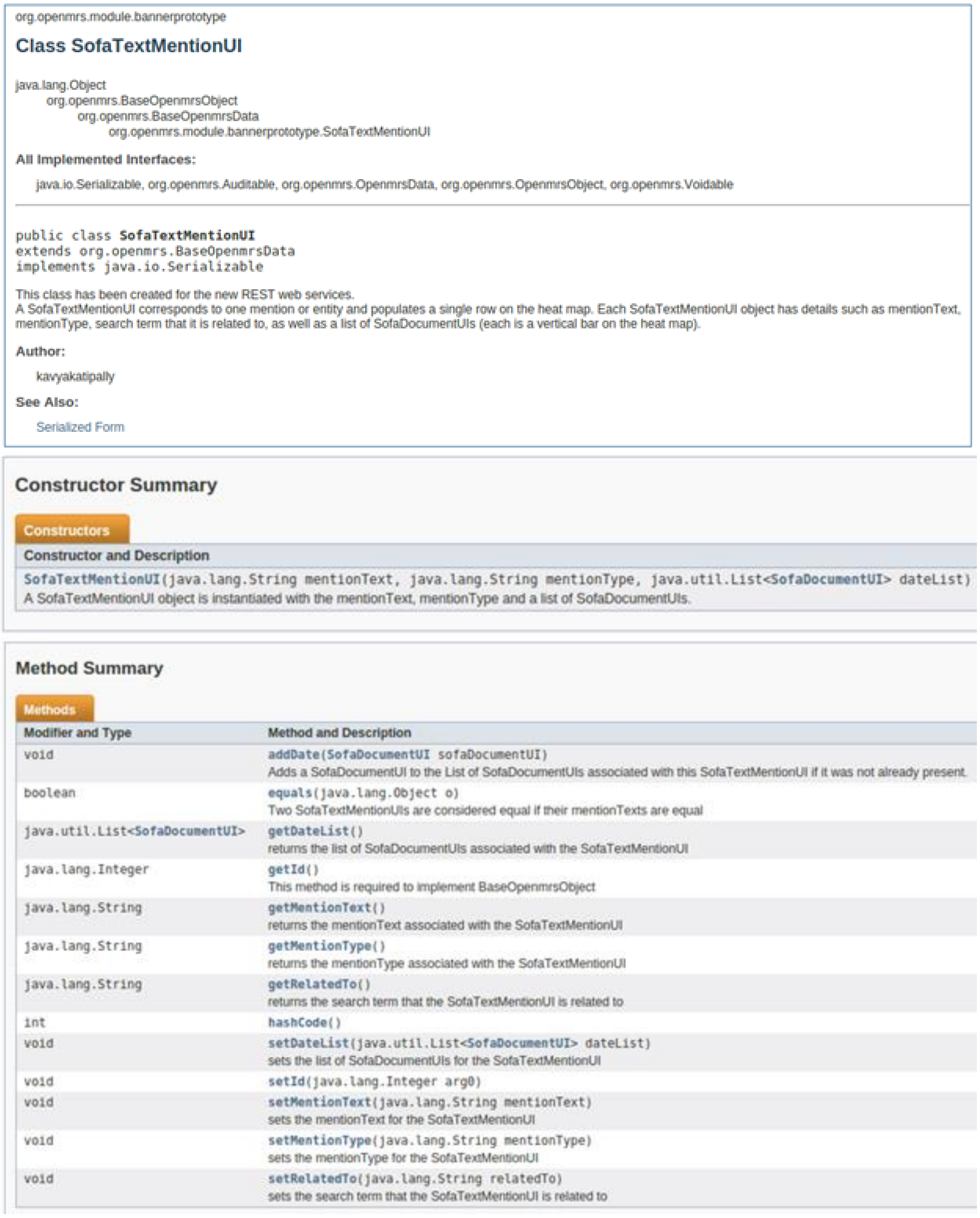
```
getChangedBy, getCreator, getDateChanged, getDateVoided, getVoided, getVoidedBy, getVoidReason, setChangedBy, setCreator,
setDateChanged, setDateVoided, setVoided, setVoidedBy, setVoidReason
```

Methods inherited from class org.openmrs.BaseOpenmrsObject

```
equals, hashCode, toString
```

Figure 16: Javadoc excerpts for SofaDocumentUI class

3.2.8 SofaTextMentionUI



org.openmrs.module.bannerprototype

Class SofaTextMentionUI

java.lang.Object
    org.openmrs.BaseOpenmrsObject
        org.openmrs.BaseOpenmrsData
            org.openmrs.module.bannerprototype.SofaTextMentionUI

All Implemented Interfaces:

java.io.Serializable, org.openmrs.Auditable, org.openmrs.OpenmrsData, org.openmrs.OpenmrsObject, org.openmrs.Voidable

```
public class SofaTextMentionUI
extends org.openmrs.BaseOpenmrsData
implements java.io.Serializable
```

This class has been created for the new REST web services.
A SofaTextMentionUI corresponds to one mention or entity and populates a single row on the heat map. Each SofaTextMentionUI object has details such as mentionText, mentionType, search term that it is related to, as well as a list of SofaDocumentUIs (each is a vertical bar on the heat map).

Author:

kavyakatipally

See Also:

Serialized Form

Constructor Summary

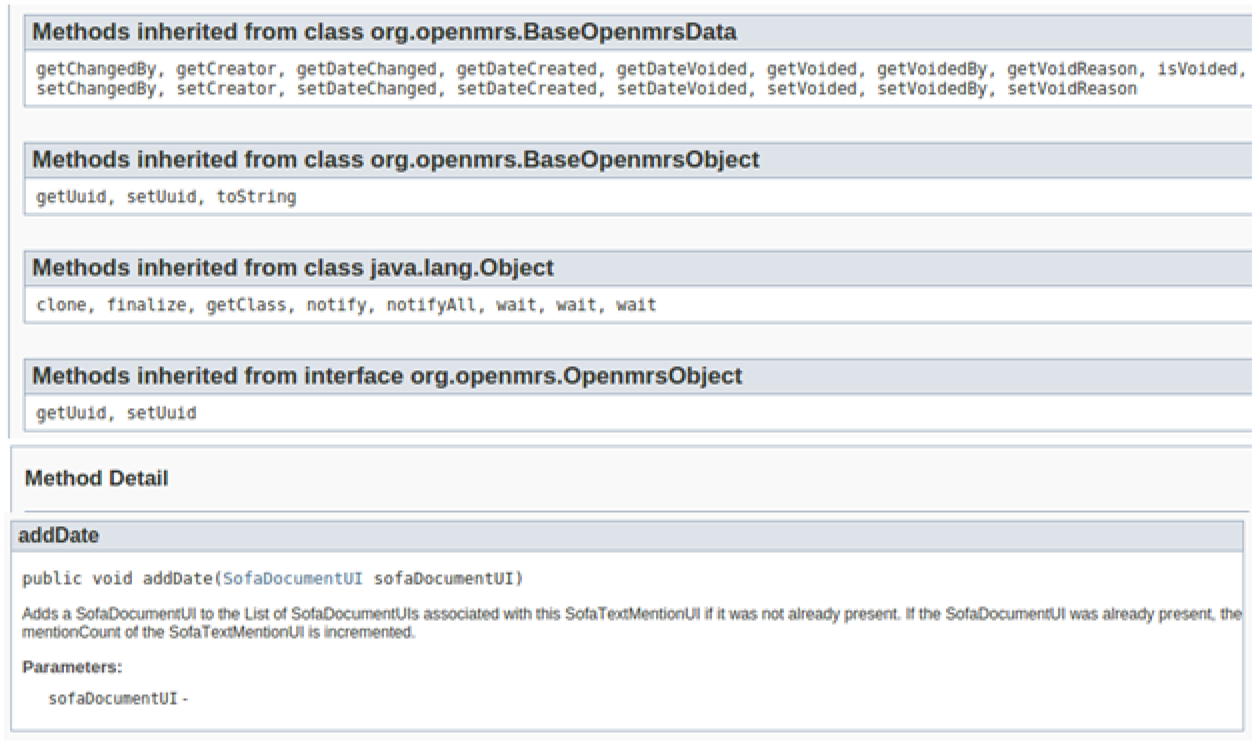
Constructors

Constructor and Description
`SofaTextMentionUI(java.lang.String mentionText, java.lang.String mentionType, java.util.List<SofaDocumentUI> dateList)` A SofaTextMentionUI object is instantiated with the mentionText, mentionType and a list of SofaDocumentUIs.

Method Summary

Methods

Modifier and Type	Method and Description
`void`	`addDate(SofaDocumentUI sofaDocumentUI)` Adds a SofaDocumentUI to the List of SofaDocumentUIs associated with this SofaTextMentionUI if it was not already present.
`boolean`	`equals(java.lang.Object o)` Two SofaTextMentionUIs are considered equal if their mentionTexts are equal
`java.util.List<SofaDocumentUI>`	`getDateList()` returns the list of SofaDocumentUIs associated with the SofaTextMentionUI
`java.lang.Integer`	`getId()` This method is required to implement BaseOpenmrsObject
`java.lang.String`	`getMentionText()` returns the mentionText associated with the SofaTextMentionUI
`java.lang.String`	`getMentionType()` returns the mentionType associated with the SofaTextMentionUI
`java.lang.String`	`getRelatedTo()` returns the search term that the SofaTextMentionUI is related to
`int`	`hashCode()`
`void`	`setDateList(java.util.List<SofaDocumentUI> dateList)` sets the list of SofaDocumentUIs for the SofaTextMentionUI
`void`	`setId(java.lang.Integer arg0)`
`void`	`setMentionText(java.lang.String mentionText)` sets the mentionText for the SofaTextMentionUI
`void`	`setMentionType(java.lang.String mentionType)` sets the mentionType for the SofaTextMentionUI
`void`	`setRelatedTo(java.lang.String relatedTo)` sets the search term that the SofaTextMentionUI is related to


Methods inherited from class org.openmrs.BaseOpenmrsData

```
getChangedBy, getCreator, getDateChanged, getDateCreated, getDateVoided, getVoided, getVoidedBy, getVoidReason, isVoided,
setChangedBy, setCreator, setDateChanged, setDateCreated, setDateVoided, setVoided, setVoidedBy, setVoidReason
```

Methods inherited from class org.openmrs.BaseOpenmrsObject

```
getUuid, setUuid, toString
```

Methods inherited from class java.lang.Object

```
clone, finalize, getClass, notify, notifyAll, wait, wait, wait
```

Methods inherited from interface org.openmrs.OpenmrsObject

```
getUuid, setUuid
```

Method Detail

addDate

```
public void addDate(SofaDocumentUI sofaDocumentUI)
```

Adds a SofaDocumentUI to the List of SofaDocumentUIs associated with this SofaTextMentionUI if it was not already present. If the SofaDocumentUI was already present, the mentionCount of the SofaTextMentionUI is incremented.

Parameters:

`sofaDocumentUI` -

Figure 17: Javadoc excerpts for SofaTextMentionUI class

4. REST web services

Four REST resources have been added to the new version of Visit Notes Analysis module.

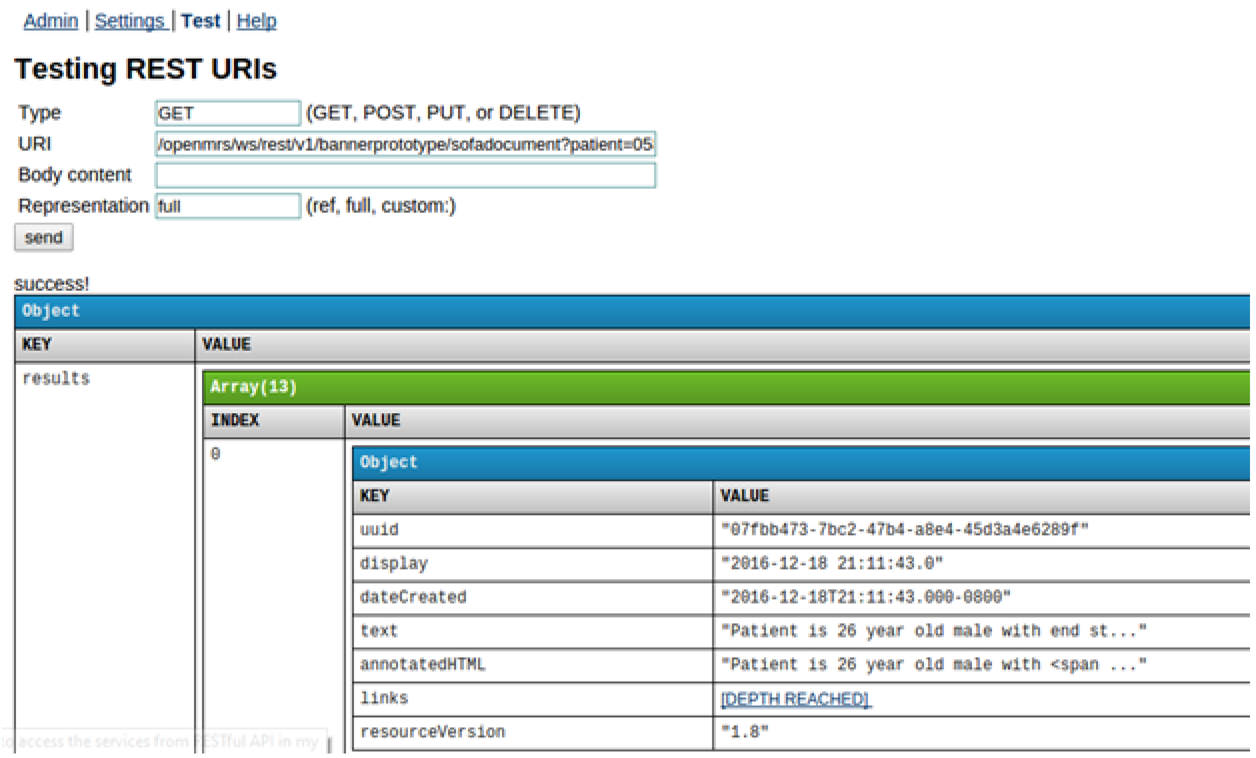
4.1 SofaDocument Resource

The SofaDocument Resource is used to populate two sections of the UI. On page 1, it is used to populate the D3 visualization of all visit notes for a given patient. An example of the REST call:

/openmrs/ws/rest/v1/bannerprototype/sofadocument?patient=0586cbb8-56f1-4621-9ea6-4d53cb44884c

Here, all visit dates are retrieved for patient with id: 0586cbb8-56f1-4621-9ea6-4d53cb44884c

A screenshot of the test page for this REST call is displayed:



Testing REST URIs

Type	GET (GET, POST, PUT, or DELETE)
URI	/openmrs/ws/rest/v1/bannerprototype/sofadocument?patient=05
Body content	
Representation	full (ref, full, custom:)

```
success!
Object
KEY       VALUE
results   Array(13)
          INDEX  VALUE
          0      Object
                 KEY              VALUE
                 uuid             "07fbb473-7bc2-47b4-a8e4-45d3a4e6289f"
                 display          "2016-12-18 21:11:43.0"
                 dateCreated      "2016-12-18T21:11:43.000-0800"
                 text             "Patient is 26 year old male with end st..."
                 annotatedHTML    "Patient is 26 year old male with <span ..."
                 links            [DEPTH REACHED]
                 resourceVersion  "1.8"
```

Figure 18: REST test page – Get all SofaDocuments for a given patient

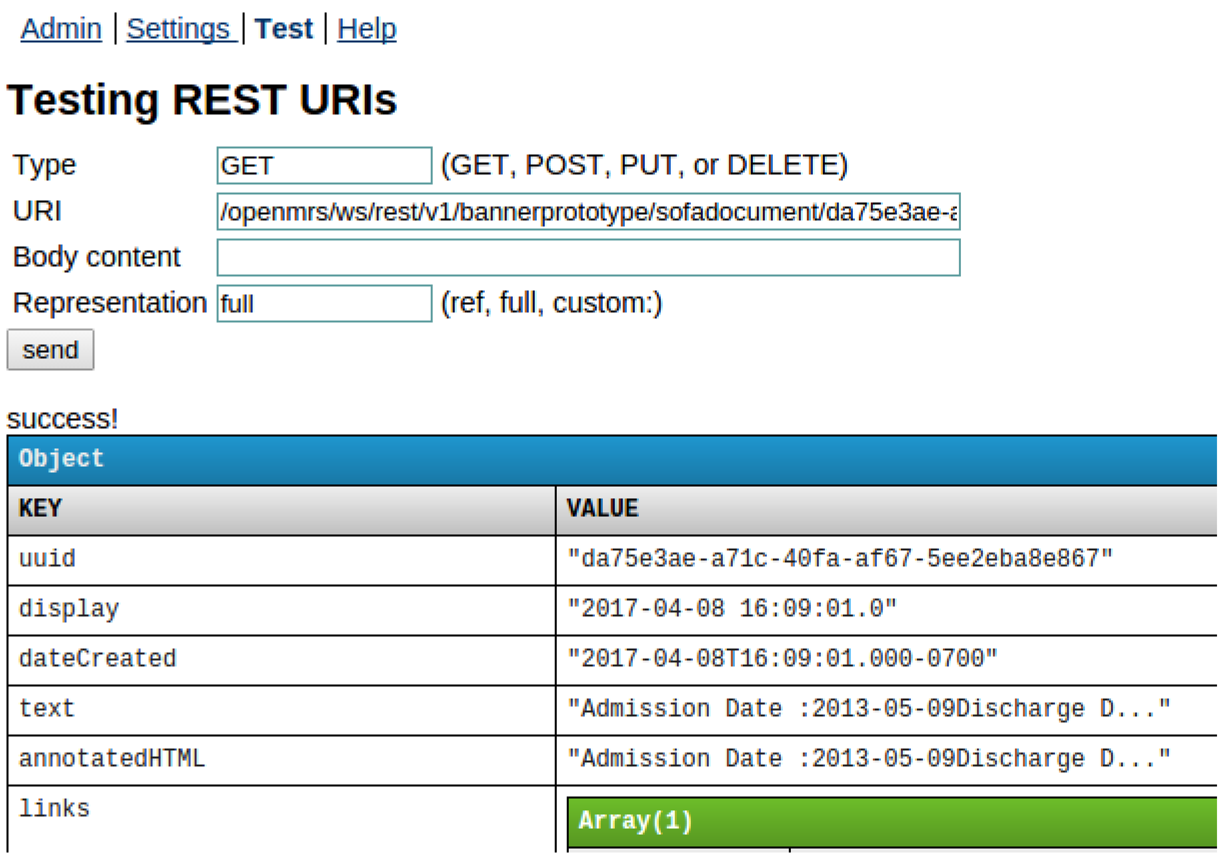
The SofadocumentResource calls NLPService's getSofaDocumentsByPatient function, which in turn calls the dao.getSofaDocumentsByPatient function. Code from SofadocumentResource:

```
@Override
	protected PageableResult doSearch(RequestContext context) {

		Patient patient = context.getParameter("patient") != null ?
Context.getPatientService().getPatientByUuid(
		    context.getParameter("patient")) : null;

		return new
NeedsPaging<SofaDocument>(Context.getService(NLPService.class).getSofaDocumentsByPati
ent(patient),
		       context);

	}
```

The SofaDocument Resource is also used to populate Page 2's Visit Note Rendering section. An example REST call which specifies the UUID of the SofaDocument for which the text is to be retrieved:

/openmrs/ws/rest/v1/bannerprototype/sofadocument/da75e3ae-a71c-40fa-af67-5ee2eba8e867

A screenshot of the test page for this REST call:


Admin | Settings | **Test** | Help

Testing REST URIs

Type	GET (GET, POST, PUT, or DELETE)
URI	/openmrs/ws/rest/v1/bannerprototype/sofadocument/da75e3ae-a
Body content	
Representation	full (ref, full, custom:)

send

success!

Object

KEY	VALUE
uuid	"da75e3ae-a71c-40fa-af67-5ee2eba8e867"
display	"2017-04-08 16:09:01.0"
dateCreated	"2017-04-08T16:09:01.000-0700"
text	"Admission Date :2013-05-09Discharge D..."
annotatedHTML	"Admission Date :2013-05-09Discharge D..."
links	Array(1)

Figure 19: REST test page – Get SofaDocument for a given UUID

The SofadocumentResource calls NLPService's getSofaDocumentByUuid function, which in turn calls the DAO's getSofaDocumentByUuid function.

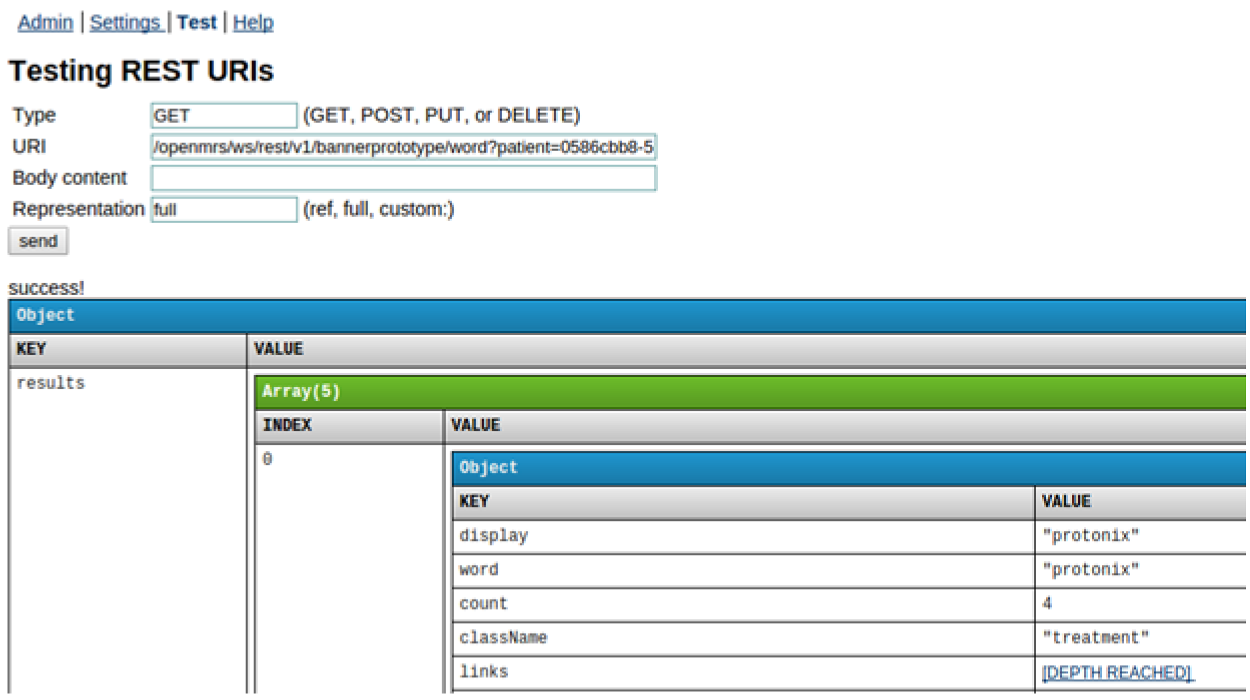
```
@Override
	public SofaDocument getByUniqueId(String uuid) {
		return Context.getService(NLPService.class).getSofaDocumentByUuid(uuid);
	}
```

4.2 Word Resource

The WordResource is used to populate the word cloud on page 1 based on constraints such as the start and end dates, the patient ID, the entity type selected and the number of terms to display in the word cloud. An example REST call is:

/openmrs/ws/rest/v1/bannerprototype/word?patient=0586cbb8-56f1-4621-9ea6-4d53cb44884c&startDate=2015-05-07&endDate=2017-05-07&entityType=All&numTerms=5

Here is a screenshot of the REST test page for this end point:



Admin | Settings | **Test** | Help

Testing REST URIs

Type	GET	(GET, POST, PUT, or DELETE)
URI	/openmrs/ws/rest/v1/bannerprototype/word?patient=0586cbb8-5	
Body content		
Representation	full	(ref, full, custom:)

send

success!

```
Object
KEY       VALUE
results   Array(5)
          INDEX  VALUE
          0      Object
                 KEY         VALUE
                 display     "protonix"
                 word        "protonix"
                 count       4
                 className   "treatment"
                 links       [DEPTH REACHED]
```

Figure 20: REST test page – Get words for a given patient, start and end dates, entityType, numOfTerms

WordResource calls NLPService's getSofaDocumentsByPatientAndDateRange method to get all SofaDocuments for a given patient, start and end dates. Depending upon the entity type specified by the user, all SofaTextMentions are added to a Word Cloud and the top SofaTextMentions are retrieved, limited by the number of terms specified by the user. Here's the code from the WordResource class:

```
@Override
	protected PageableResult doSearch(RequestContext context) {

		Patient patient = context.getParameter("patient") != null ?
Context.getPatientService().getPatientByUuid(
				context.getParameter("patient")) : null;

		Date startDate = context.getParameter("startDate") != null ? (Date)
ConversionUtil.convert(
				context.getParameter("startDate"), Date.class) : null;
		Date endDate = context.getParameter("endDate") != null ? (Date)
ConversionUtil.convert(
		    context.getParameter("endDate"), Date.class) : null;

		String entityType = context.getParameter("entityType");
		Integer numTerms = Integer.parseInt(context.getParameter("numTerms"));

		WordCloud wordcloud = new WordCloud();

		List<SofaDocument> allSofaDocuments =
Context.getService(NLPService.class).getSofaDocumentsByPatientAndDateRange(
		    patient, startDate, endDate);

		if (entityType.equals("Problems")) {
			for (SofaDocument sd : allSofaDocuments) {
				addToCloud(wordcloud, sd.getProblemMentions());
			}
		} else if (entityType.equals("Treatments")) {
			for (SofaDocument sd : allSofaDocuments) {
				addToCloud(wordcloud, sd.getTreatmentMentions());
			}
		} else if (entityType.equals("Tests")) {
			for (SofaDocument sd : allSofaDocuments) {
				addToCloud(wordcloud, sd.getTestMentions());
			}
		} else {
			for (SofaDocument sd : allSofaDocuments) {
				addToCloud(wordcloud, sd.getAllMentions());
			}
		}

		List<Word> wordList = wordcloud.getTopWordsShuffled(numTerms);
		return new NeedsPaging<Word>(wordList, context);
	}
```

4.3 SofaTextMentionUI Resource

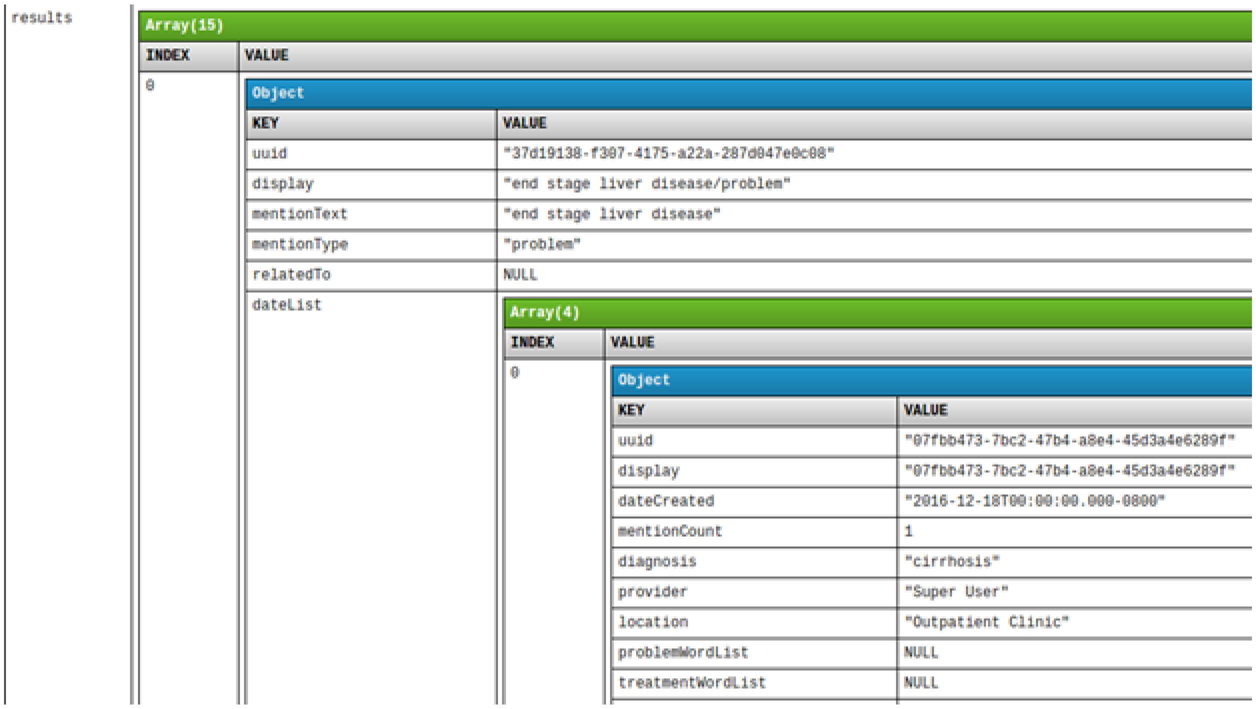
For page 2's heat map and Visit Note List sections, two new classes were created. Javadocs for these classes were shared in the previous section. One is the SofaDocumentUI, which is analogous to a vertical bar on the heat map corresponding to a single note. It has the instance variables – UUID, patient, dateCreated, mentionCount, diagnosis, provider, location, problemWordList, treatmentWordList and testWordList. The diagnosis, provider, location variables are used to populate the tooltip on the heat map for the particular note. The mentionCount is used to show the frequency of occurrence of the mention in this note. The problemWordList, treatmentWordList and testWordList are kept empty for the heat map since these lists of all mentions are not required on the heat map. They are required in the Visit Note List section for the filter functionality.

The SofaTextMentionUI object is analogous to a single search term row in the heat map. It has these instance variables – mentionText, mentionType, relatedTo and dateList. The relatedTo variable is used to identify the search term to which a term is related, and the dateList variable stores a List of SofaDocumentUI objects for each SofaTextMentionUI object. These SofaTextMentionUI objects are used to populate the heat map visualization.

The SofaTextMentionUI REST Resource is used to populate the heat map for given search terms, start and end dates and patient ID. A sample REST call is -

/openmrs/ws/rest/v1/bannerprototype/sofatextmentionui?patient=0586cbb8-56f1-4621-9ea6-4d53cb44884c&startDate=2015-04-30&endDate=2017-04-30&searchTerms=end stage liver disease

The test page shows an array of SofaTextMentionUI objects with each one mapping to a dateList, which is in turn an array of SofaDocumentUI objects.



```
results   Array(15)
          INDEX   VALUE
          0       Object
                  KEY           VALUE
                  uuid          "37d19138-f307-4175-a22a-287d047e0c08"
                  display       "end stage liver disease/problem"
                  mentionText   "end stage liver disease"
                  mentionType   "problem"
                  relatedTo     NULL
                  dateList      Array(4)
                                INDEX   VALUE
                                0       Object
                                        KEY                 VALUE
                                        uuid                "07fbb473-7bc2-47b4-a8e4-45d3a4e6289f"
                                        display             "07fbb473-7bc2-47b4-a8e4-45d3a4e6289f"
                                        dateCreated         "2016-12-18T00:00:00.000-0800"
                                        mentionCount        1
                                        diagnosis           "cirrhosis"
                                        provider            "Super User"
                                        location            "Outpatient Clinic"
                                        problemWordList     NULL
                                        treatmentWordList   NULL
```

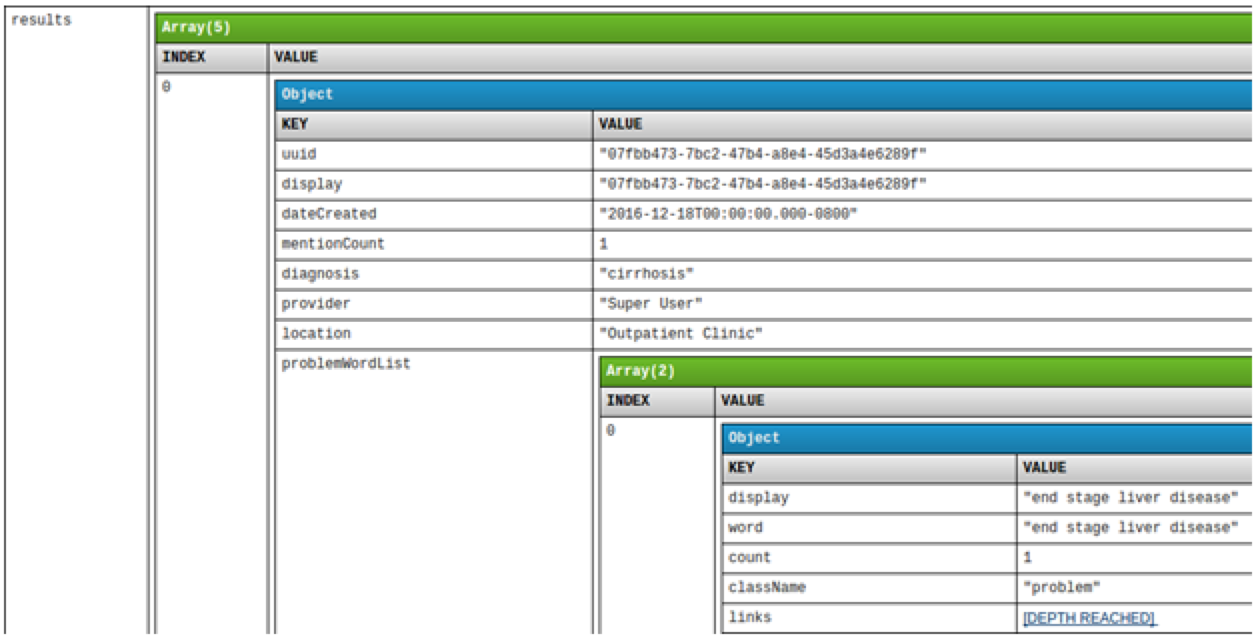
Figure 21: REST test page – Get SofaTextMentionUIs for a given patient, start &end dates, search terms

The SofaTextMentionUI Resource calls NLPService's getSofaTextMentionUIsByConstraints() method. The NLPServiceImpl has the business logic for how the heat map gets populated. When the search terms exceed the limit (for example, 3), it calls the dao.getSofaTextMentionUIsByConstraints(patient, startDate, endDate, searchTerms) method directly to get the SofaTextMentionUIs to populate the heat map. On the other hand, if the number of search terms is less than the limit, related terms are to be displayed on the UI. SofaDocuments for each search term are first retrieved by calling dao.getSofaDocumentsByConstraints(patient, startDate, endDate, term). The most frequent 5 problems, 5 treatments and 5 tests are retrieved from all SofaDocuments corresponding to a search term. Finally, the SofaTextMentionUIs which populate each row in the heat map are obtained by calling dao.getSofaTextMentionUIsByConstraints(patient, startDate, newEndDate, allTopTermsArr).

In the code for dao.getSofaTextMentionUIsByConstraints – First, a database query is run which retrieves the SofaTextMention's text and type along with all the SofaDocuments in which the mention occurs, and the UUIDs, dateCreated and encounter IDs for each SofaDocument. Using the encounter ID, we get the Provider, Location and Diagnosis details for each SofaDocument. A SofaDocumentUI object is instantiated for each SofaDocument and added to an ArrayList. For each mention, a SofaTextMentionUI object is instantiated and the dateList is set. Finally, an ArrayList of SofaTextMentionUIs is returned.

4.4 SofaDocumentUI Resource

The SofaDocumentUI Resource is used to make two REST calls – one to populate the Visit Note List for given search terms and another to populate the Visit Note List for a particular visit date.
An example of the first REST call to populate the Visit Note List for given search terms:
/openmrs/ws/rest/v1/bannerprototype/sofadocumentui?patient=0586cbb8-56f1-4621-9ea6-4d53cb44884c&startDate=2015-05-03&endDate=2017-05-03&searchTerms=end stage liver disease&searchTerms=protonix
Here's a screenshot of the REST test page which shows an array of SofaDocumentUIs, with the problemWordList populated. The problem, treatment and test wordlists are populated for each SofaDocumentUI object so that filtering by any problem, treatment or test entity can work on the Visit Note List.



```
results   Array(5)
          INDEX  VALUE
          0      Object
                 KEY              VALUE
                 uuid             "07fbb473-7bc2-47b4-a8e4-45d3a4e6289f"
                 display          "07fbb473-7bc2-47b4-a8e4-45d3a4e6289f"
                 dateCreated      "2016-12-18T00:00:00.000-0800"
                 mentionCount     1
                 diagnosis        "cirrhosis"
                 provider         "Super User"
                 location         "Outpatient Clinic"
                 problemWordList  Array(2)
                                  INDEX  VALUE
                                  0      Object
                                         KEY        VALUE
                                         display    "end stage liver disease"
                                         word       "end stage liver disease"
                                         count      1
                                         className  "problem"
                                         links      [DEPTH REACHED]
```

Figure 22: REST test page – Get SofaDocumentUIs for a given patient, start & end dates, search terms

The SofaDocumentUI Resource calls NLPService's getSofaDocumentUIsByConstraints () method. The NLPServiceImpl has the business logic for how the Visit Note List gets populated, which is similar to the logic in the previous section. When the search terms exceed the limit (for example, 3), it calls the dao.getSofaDocumentUIsByConstraints(patient, startDate, newEndDate, searchTerms) method directly to get the SofaDocumentUIs to populate the Visit Note List. On the other hand, if the number of search terms is less than the limit, dates for related terms are also to be displayed on the UI. SofaDocuments for each search term are first retrieved by calling dao.getSofaDocumentsByConstraints(patient, startDate, newEndDate, term). The most frequent 5 problems, 5 treatments and 5 tests are retrieved from all SofaDocuments corresponding to a search term. Finally, the SofaDocumentUIs which populate each row in the Visit Note List are obtained by calling dao.getSofaDocumentUIsByConstraints(patient, startDate, newEndDate, allTopTermsArr).

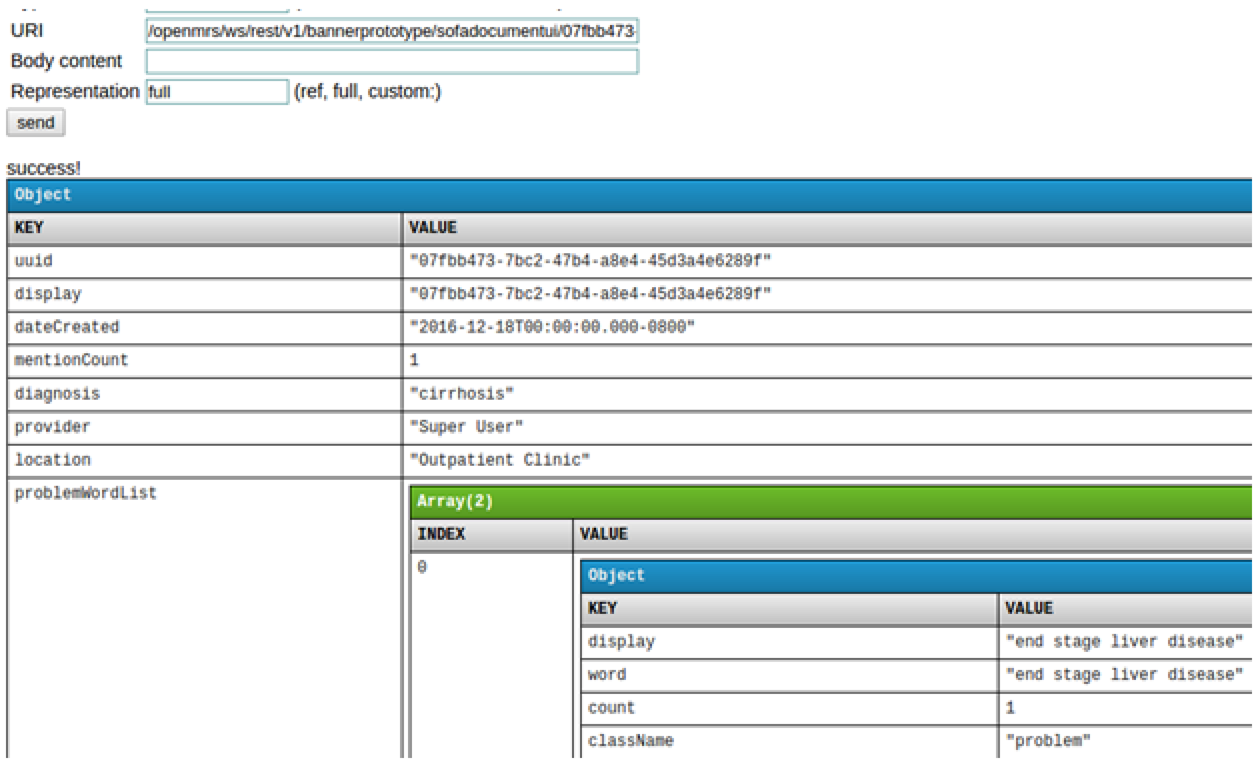
In the code for dao.getSofaDocumentUIsByConstraints – First, a database query is run which retrieves the UUIDs, dateCreated and encounter IDs for distinct SofaDocuments in which the mentions occur. Using the encounter ID, we get the Provider, Location and Diagnosis details for each SofaDocument.

Lists of Problems, Treatments and Tests that occur in the SofaDocument are added to the problemWordList, treatmentWordList and testWordList respectively. A SofaDocumentUI object is instantiated for each SofaDocument and an ArrayList of SofaDocumentUIs is returned.

An example of the second REST end point available on the SofaDocumentUI resource, which can be used to populate the Visit Note List for a particular visit date (please see Figure 4.5):

/openmrs/ws/rest/v1/bannerprototype/sofadocumentui/07fbb473-7bc2-47b4-a8e4-45d3a4e6289f

A screenshot of the REST test page for this call shows a single SofaDocumentUI object with the problemWordList, treatmentWordList and testWordList populated.



URI	/openmrs/ws/rest/v1/bannerprototype/sofadocumentui/07fbb473
Body content	
Representation	full (ref, full, custom:)

send

success!

Object

KEY	VALUE
uuid	"07fbb473-7bc2-47b4-a8e4-45d3a4e6289f"
display	"07fbb473-7bc2-47b4-a8e4-45d3a4e6289f"
dateCreated	"2016-12-18T00:00:00.000-0800"
mentionCount	1
diagnosis	"cirrhosis"
provider	"Super User"
location	"Outpatient Clinic"
problemWordList	Array(2)

problemWordList – Array(2)

INDEX	VALUE
0	Object

Object

KEY	VALUE
display	"end stage liver disease"
word	"end stage liver disease"
count	1
className	"problem"

Figure 23: REST test page – Get SofaDocumentUI for a given SofaDocument UUID

The SofaDocumentUI resource calls NLPService's getSofaDocumentUIBySofaDocUuid() method which calls dao.getSofaDocumentUIBySofaDocUuid(). This method queries the database for the dateCreated and encounter ID of the SofaDocument with the UUID of the date clicked on page 1. Using the encounter ID, we get the Provider, Location and Diagnosis details for the SofaDocument. Lists of Problems, Treatments and Tests that occur in the SofaDocument are added to the problemWordList, treatmentWordList and testWordList respectively. A SofaDocumentUI object is instantiated and returned.

Code from the SofaDocumentUI resource is shown below:

```
@Override
	public SofaDocumentUI getByUniqueId(String sofaDocUuid) {
		return
Context.getService(NLPService.class).getSofaDocumentUIBySofaDocUuid(sofaDocUuid);

	}
```

5. User interface development

5.1 AngularJS set up

notesNLPng.gsp is set as the view and NotesNLPngPageController is the corresponding controller as part of the Spring MVC set up in this module. Instead of rendering the GSP server-side, an AngularJS app is embedded in this GSP. The code from notesNLPng.gsp is shown here:

```
<body ng-app="visitNotesApp">
      <base href="/"/>
      <div ng-view></div>
 </body>
```

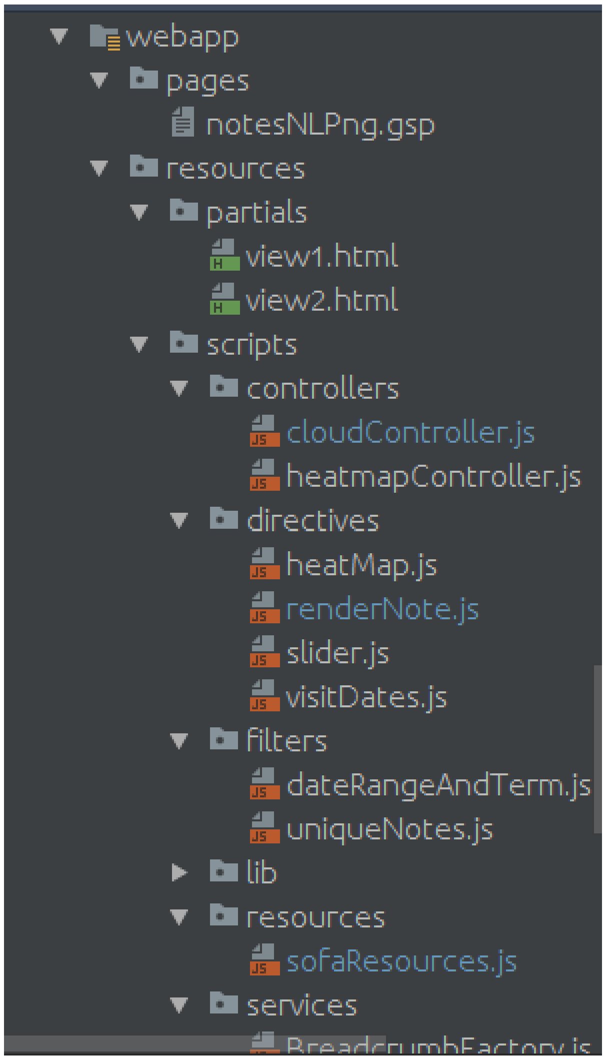
The ng-app directive specifies the root element of the AngularJS application. The ng-view directive specifies the location where partial views are to be rendered.

We want to navigate to two separate views, but keep this as a single page application with no page re-loading, so we use Angular’s ngRoute module. $routeProvider is defined using the config method and it is used to configure different routes in this application. The default route is also set using the otherwise() method. Here is a code snippet from app.js:

```
      var visitNotesApp = angular.module('visitNotesApp', [
        'ngRoute', 'ngResource', 'ngAnimate', 'ngSanitize', 'ui.bootstrap'
      ])
      .config(['$routeProvider', function($routeProvider) {
            $routeProvider.when('/view1', {
                  templateUrl: '/' + OPENMRS_CONTEXT_PATH +
      '/ms/uiframework/resource/bannerprototype/partials/view1.html',
                  css: '/' + OPENMRS_CONTEXT_PATH +
      '/ms/uiframework/resource/bannerprototype/styles/app.css'
        })
            $routeProvider.when('/view2', {
                  templateUrl: '/' + OPENMRS_CONTEXT_PATH +
      '/ms/uiframework/resource/bannerprototype/partials/view2.html',
                  css: '/' + OPENMRS_CONTEXT_PATH +
      '/ms/uiframework/resource/bannerprototype/styles/app.css'
        })
            $routeProvider.otherwise({redirectTo: '/view1'});
      }]);
```

5.2 AngularJS folder structure

Here is the folder structure of the AngularJS files – view1.html and view2.html under webapp/resources/partials are the html files for page 1 and page 2 of the application respectively. The webapp/resources/scripts folder has separate folders for AngularJS controllers, directives, filters, resources, services etc.



```
▼ webapp
  ▼ pages
      notesNLPng.gsp
  ▼ resources
    ▼ partials
        view1.html
        view2.html
    ▼ scripts
      ▼ controllers
          cloudController.js
          heatmapController.js
      ▼ directives
          heatMap.js
          renderNote.js
          slider.js
          visitDates.js
      ▼ filters
          dateRangeAndTerm.js
          uniqueNotes.js
      ▶ lib
      ▼ resources
          sofaResources.js
      ▼ services
          BreadcrumbFactory.js
```

Figure 24: AngularJS folder structure

5.3 AngularJS controllers

Angular controllers are used to set up the initial state of the $scope object and to add properties and methods to the $scope. They contain business logic. There is a single controller for page 1 called cloudController.js and one controller for page 2 called heatmapController.js.

Examples of methods in the cloud controller are addToSearch() and page1Submit(). addToSearch() is called when any term in the word cloud is clicked, and page1Submit() is called when the form at the bottom of page 1 is submitted. The relevant code snippets are shown from view1.html and cloudController.js:

```
<a ngHref="#" ng-click="addToSearch(term.word)">
<form class="searchbottom" ng-submit="page1Submit(searchInput)">

$scope.addToSearch = function(name){
			if($scope.searchInput === "")
				$scope.searchInput += name ;
			else
				$scope.searchInput += ", " + name ;
		};

		$scope.page1Submit = function(searchInput){
			SearchFactory.setSearchTerms($scope.searchInput);
			$location.url('/view2');
		};
```

The controllers also consume JSON data from REST web services. This is discussed further in the next section. In addition, the controllers use $scope.$watch to register a listener callback to be executed whenever the watch expression changes. A code snippet below from cloudController.js shows the use of $watch. The slider min and max dates, entity type and number of terms to display are tracked, and a REST web service call for the cloud data is made if any of the above properties change.

```
$scope.$watch('[sliderMinDate, sliderMaxDate, entityType, displayNumTerm]',
               function(newVals, oldVals) {
                         $scope.words = WordResources.displayCloud({
```

The heatmapController handles the business logic for how page 2 gets populated. If both the search terms and the visit date selected on page 1 are “falsy”, the application is reloaded by re-routing to page 1. On the other hand, if search terms were entered on page 1, the heat map and visit note list are populated with data returned from corresponding REST calls. If a visit date was clicked on page 1, the visit note list and rendering sections are updated with data for the particular date. The code is shown:

```
if (!$scope.searchInput && !$scope.visitDateUuid) {
      $location.url('/view1');
} else if ($scope.searchInput){
//populate the heat map and visit note list with data for the search terms
} else if ($scope.visitDateUuid) {
//populate the visit note list and rendering with data for the date selected
}
```

In the heatmapController.js, the business logic to handle these three scenarios is similar – loading page 2 with search terms entered on page 1, search terms and dates submitted on page 2 and breadcrumbs clicked on page 2. All of these cases result in re-loading the heat map and visit note list sections based on the search terms and date range entered.

5.4 AngularJS directives and D3.js

In AngularJS, directives are used for DOM manipulation. Angular comes with a set of built-in directives such as ngModel, ngClass and ngBind. This application creates four custom directives – slider.js for the slider on page 1, visitDates.js for the D3.js visit dates visualization at the top of page 1, heatMap.js for the D3.js heat map visualization on page 2 and renderNote.js for the Visit Note Rendering section on page 2.

Directives are matched based on element names (E), attributes (A), class names (C) and comments (M). A directive can specify which of the four matching types it supports in the restrict property of the directive definition object. Isolate scope has been used for the custom directives, so the scope inside each directive is separated from the outside scope. The outside scope is then mapped by binding data to the directive's isolate scope. Here's a code snippet from the visitDates directive with five properties passed to the isolate scope:

```
visitNotesApp.directive('visitDates', function($compile){
        return {
        restrict: 'E',
        scope: {
                visitDatesData: '=visitDatesData',
                visitDatesDataUpdated: '=visitDatesDataUpdated',
                visitDateUuid: '=visitDateUuid',
                sliderMinDate: '=sliderMinDate',
                sliderMaxDate: '=sliderMaxDate'
        },
        link: function(scope, element, attrs, controller) {
```

D3 visualizations with SVGs are created in both the heatMap.js and visitDates.js directives. For the visitDates visualization, an SVG is first created, and then vertical bars are added for each date in the data. An X-axis is added and tooltips are added to show details upon hovering over a vertical bar. Scope.$watch is added to build the visualization again if the slider min and max dates have changed or if the data has been loaded. The D3.js code below shows how a vertical bar of height '20', width '2' is added at the corresponding X-axis location for each date in the data:

```
var dates = svg.selectAll('.dates')
                .data(data)
                .enter().append('rect')
                .attr('x', function(d){ return xScale(new Date(d.dateCreated)) })
                .attr('y', 10)
                .attr("rx", 0)
                .attr("ry", 0)
                .attr('width', 2)
                .attr('height', 20)
```

The heatMap.js directive is more complex than visitDates. Properties shared with the isolate scope of the heatMap directive include the data, start and end dates for the timeline and resetMap. Other shared properties are used to populate the search bar upon right clicking a heat map term (searchInput) and to filter the Visit Note List (filterFromDate, filterToDate, matchTerm, visitListInput). HeatMap.js updates the visualization by keeping track of the current state of the heat map. The current state includes details such as entities that have been removed by the user by clicking 'X', and entities that have been toggled by clicking (+).

In heatMap.js, an SVG is first added and then entity frequencies in various rectangles are computed so that the color scale can be set. For the color scale, a frequency of '0' corresponds to a very light color, and the highest computed frequency in a rectangle corresponds to a deep red color. This color scale is applied to both the rectangles and the vertical bars and it gets re-computed with every update. Here's the code for the color scale:

```
var colorScale = d3.scaleLinear()
                    .domain([0, d3.max(AllRect, function(d){ return d.totFreq;})])
                       .range(['#ffffd9', '#ff0033']);
```

The heat map is then populated row by row for each entity. In each row, (+) or (-) is displayed for search terms, then the entity name is displayed. The entity can be right clicked to be added to the search bar, and left clicked to be added to the filter on the Visit Note List section. The heat map rectangles and vertical bars are added, which display tooltips upon hovering and filter the Visit Note list upon clicking. Finally the 'X' symbol on each row is displayed. Below all the entities, the X-axis and a color scale legend are added. There is a scope.$watch to track the input data and the resetMap variable, which updates the visualization if these values change. Here's the code snippet for the display of an entity.

```
var yLabel = d3.select(this).append("text")
             .text(function (d) { return d.mentionText; })
             .attr("x", 1.4*termWidth)
             .attr("y", function (d, i) { return ((gridHeight*3)/4) + (j *
gridHeight); })
             .style('text-anchor', 'end')
             .style('font-family', 'sans-serif')
             .attr('class', 'yLabel')
             .each( //sets the color of the text
                    function(d){
                           d3.select(this).classed(d.mentionType, true)
                    })
             .on("contextmenu", function(d, i){ //right click to add to search bar
 d3.event.preventDefault();
 scope.$apply(function() {
                          if(scope.searchInput === "")
```

```
                                 scope.searchInput += d.mentionText ;
else
                                 scope.searchInput += ", " + d.mentionText ;
                    });
             })
             .on("click", function(d, i){ //left click to add to Visit List filter
 scope.$apply(function() {
                          scope.scrollToList = true;
                          scope.visitListInput = d.mentionText;
                   });
             });
```

For the date range slider, a library called jQRangeSlider has been used. The slider.js directive handles the bounds, default min and max dates, minimum range and updates the slider min and max dates.

5.5 AngularJS filters and services

Angular filters are used to filter data. In this application, they are used to filter the Visit Note List section. The custom dateRangeAndTerm.js filter is used to filter visit notes displayed in the table by the start and end dates and the entity selected. An in-built Angular filter is also used which filters the visit notes by the input element associated with visitListInput. This is a very useful filter since anything entered into the input box is compared to all the data associated with a visit note. For example, filtering by '2016' filters for visit notes which have '2016' anywhere in the data. Similarly, filtering by 'end stage liver disease' filters for visit notes with this term, and it is expected to be part of the problemWordList of the visit note. Here's the corresponding code snippet from view2.html:

```
<tr ng-repeat="visitNote in filteredVisitNotes = (visitNotes | orderBy: 'date' |
dateRangeAndTerm: filterFromDate : filterToDate : matchTerm | filter:visitListInput |
uniqueNotes) | limitTo:5:5*(currentPage-1)"
                        ng-click="selectNote()">
                     <td>{{visitNote.date | date:'mediumDate'}}</td>
                        <td>{{visitNote.diagnosis}}</td>
                        <td>{{visitNote.provider}}</td>
                        <td>{{visitNote.location}}</td>
                    </tr>
```

Angular services are used to share state between the two controllers used in this application.

6. Package Structure

Some directories, such as those with .js/css files have not been included.

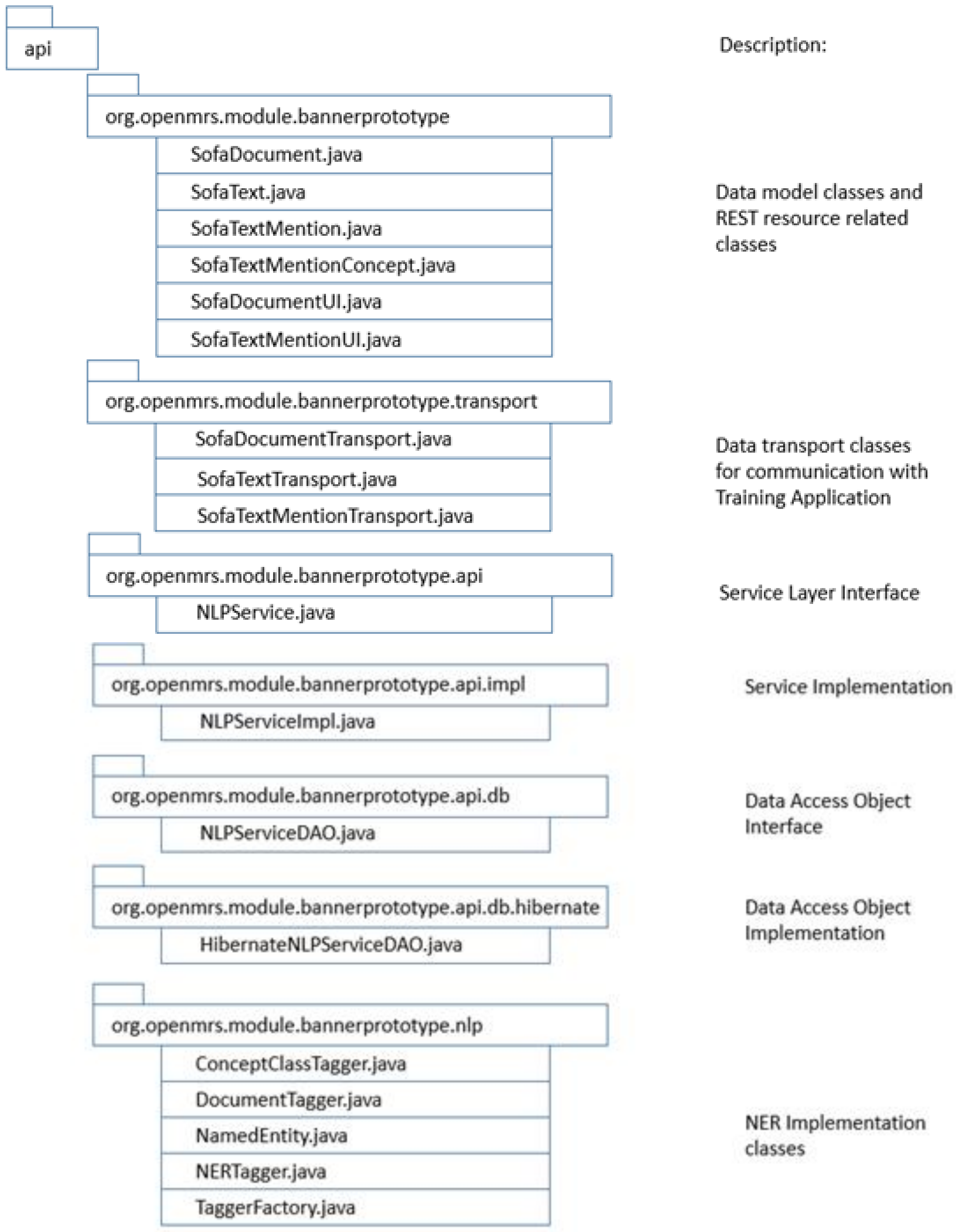

api
org.openmrs.module.bannerprototype
SofaDocument.java
SofaText.java
SofaTextMention.java
SofaTextMentionConcept.java
SofaDocumentUI.java
SofaTextMentionUI.java
org.openmrs.module.bannerprototype.transport
SofaDocumentTransport.java
SofaTextTransport.java
SofaTextMentionTransport.java
org.openmrs.module.bannerprototype.api
NLPService.java
org.openmrs.module.bannerprototype.api.impl
NLPServiceImpl.java
org.openmrs.module.bannerprototype.api.db
NLPServiceDAO.java
org.openmrs.module.bannerprototype.api.db.hibernate
HibernateNLPServiceDAO.java
org.openmrs.module.bannerprototype.nlp
ConceptClassTagger.java
DocumentTagger.java
NamedEntity.java
NERTagger.java
TaggerFactory.java
Description:
Data model classes and REST resource related classes
Data transport classes for communication with Training Application
Service Layer Interface
Service Implementation
Data Access Object Interface
Data Access Object Implementation
NER Implementation classes

Figure 25: Module API package structure

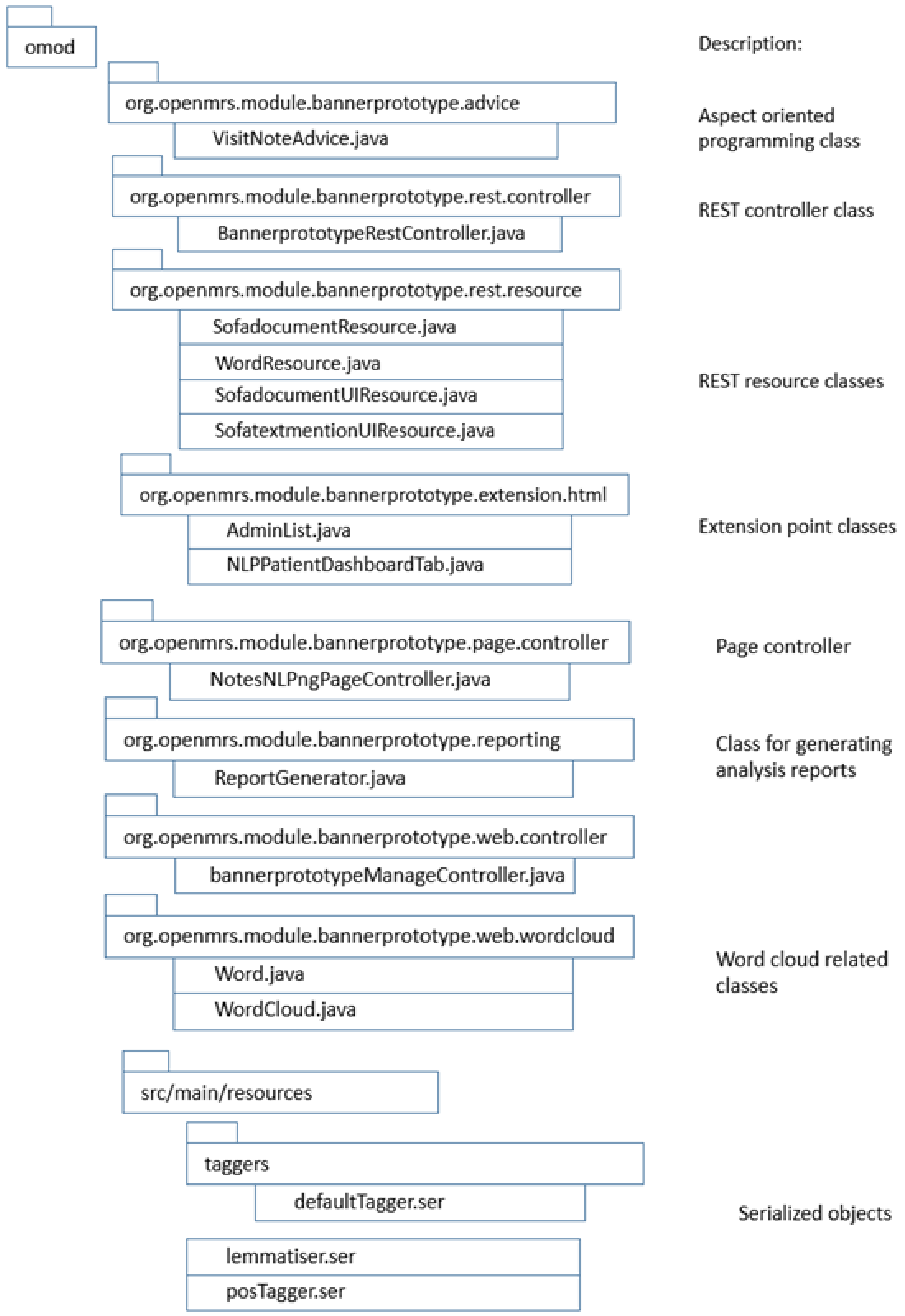

omod
org.openmrs.module.bannerprototype.advice
VisitNoteAdvice.java
org.openmrs.module.bannerprototype.rest.controller
BannerprototypeRestController.java
org.openmrs.module.bannerprototype.rest.resource
SofadocumentResource.java
WordResource.java
SofadocumentUIResource.java
SofatextmentionUIResource.java
org.openmrs.module.bannerprototype.extension.html
AdminList.java
NLPPatientDashboardTab.java
org.openmrs.module.bannerprototype.page.controller
NotesNLPngPageController.java
org.openmrs.module.bannerprototype.reporting
ReportGenerator.java
org.openmrs.module.bannerprototype.web.controller
bannerprototypeManageController.java
org.openmrs.module.bannerprototype.web.wordcloud
Word.java
WordCloud.java
src/main/resources
taggers
defaultTagger.ser
lemmatiser.ser
posTagger.ser
Description:
Aspect oriented programming class
REST controller class
REST resource classes
Extension point classes
Page controller
Class for generating analysis reports
Word cloud related classes
Serialized objects

Figure 26: Module OMOD package structure