

BODHI: A System for Crowdsourced OCR Text Correction

Student Name: Megha Gupta

IIIT-D-MTech-CS-DE-13-MT11024

Indraprastha Institute of Information Technology, New Delhi
meghag@iiitd.ac.in

Advisor(s): Dr. Haimonti Dutta and Mr. Manoj Pooleery
The Center for Computational Learning Systems (CCLS),
Columbia University, New York.
{haimonti,manoj}@ccls.columbia.edu

Submitted in partial fulfillment of the requirements
for the degree of MTech in Computer Science
with Specialization in Data Engineering

July 28, 2013

1 Abstract

Newspapers are the first draft of history – they are a rich source of information for historians, researchers, and scholars. With the advent of *digitized* newspapers, the accessibility to old historic papers has increased. The usability of archives storing these newspapers depends on the imaging technology, Optical Character Recognition (OCR) devices, zoning and segmentation, metadata extraction, searchability and web delivery systems developed to make them accessible. It is a well known fact that the OCR technology is far from perfect and often, the text generated is garbled affecting the efficiency of search and retrieval. Since the human mind excels in visual cognition and language processing tasks and machines are not able to completely recreate these skills, manual labor is involved in correcting garbled OCR.

The goal of this project is to build a system that will allow users of a digitized newspaper archive to read and correct OCR text. This report is organized as follows: section 2 and section 3 describe the functions and workflow of the system, section 4 and section 5 elaborates on its design model and system specifications; finally, implementation details with real screenshots and the challenges faced during development are mentioned in section 6.

Contents

1	Abstract	2
2	Introduction	4
2.1	Problem Description	4
2.2	Scope & Objectives	4
2.3	Software Functions	4
3	Software Project Plan	5
3.1	Workflow of the System	6
4	Design Models	7
4.1	Database Design	7
4.2	Architectural Design	9
4.2.1	Implementation	10
5	System Study	11
5.1	Hardware Specification	11
5.2	Software Specification	11
5.3	Setting up the code base	12
6	Implementation	14
6.1	Functions Implemented	14
6.2	User Interface	15
6.3	Remote Debugging Tomcat using Eclipse	15
6.4	Challenges	16
7	Conclusion	17
8	Acknowledgement	17
	Appendices	19
A	Screenshots	19

2 Introduction

2.1 Problem Description

The goal of our project is to develop a system to edit garbled OCR text from old historic newspapers. The archive under consideration is the holdings of California Digital Newspaper Collection (CDNC) [7]. The OCR software used on digital images of newspapers in this archive has generated garbled text due to poor quality of input paper¹, varying size and style of font and different column layouts. This adversely affects the retrieval effectiveness, hence the techniques for cleaning the OCR [8] need to be improvised. Often such techniques involve laborious and time consuming manual processing of data. By correcting the garbled text, the efficiency and usability of the retrieved results are expected to improve.

2.2 Scope & Objectives

The objective is to build a web interface equipped with a text correction tool for enabling the users to correct the OCR errors as they come across them. The software is expected to be usable, intuitive, simple and functions well consistently. The basic functions are:

1. Searching and displaying images of issues
2. Search by issue name and date range
3. User management functions
4. Text correction functionality

2.3 Software Functions

The project functions can be categorized into the following:

- **User management functions:** These encompass activities to manage users of the application. These include:

¹Newspapers date back to 1846

1. Registration module: Ability to enter and validate user demographic information.
2. User security module: Ability to authenticate and authorize the user. Authentication involves checking the combination of user entered login-password against an existing login-password record in the database. Authorization strategy involves accessing operations based on the role membership. Roles are used to partition the application's user base into sets of users.
3. User tracking module: Ability to track user activity, prepare statistics (how long users spent on each module, how many articles they corrected, participation in discussions)

- **Article related functions:**

1. Article search: This provides the capability to retrieve an article based on a set of criteria like article topic, date range, issue name.
2. OCR correction: This interface provide users with the capability to correct the OCR displayed along side a given article image. A basic level of authentication (based on a CAPTCHA model) is required before the user can correct the text.
3. Tag specification: This interface allows the user to specify tags corresponding to a given article. This functionality also requires a basic level of authentication.

3 Software Project Plan

This project was performed in three phases:

- **Phase 1** included laying out UI components, database connectivity, searching and displaying user requested issues with their images.
- **Phase 2** included basic functionality text correction tool, user management functions.
- **Phase 3** included advance searching and functionality of tool.

3.1 Workflow of the System

- In Phase 1, the first task was to design and build the layout of the interface. The user interface components in Vaadin can roughly be divided in two groups: components that the user can interact with and layout components for placing the other components to specific places in the UI. The development started by creating a content layout for the UI, and then the other layout components were added hierarchically, and finally the interaction components were added as the leaves of the component tree.

The root layout of this project is the Vertical Layout which holds two more Horizontal Layouts, called "mainLayout" and "bottomLayout". MainLayout contains the UI for search functionality whereas bottomLayout is divided into three panels called tool panel, image panel and function panel at left, center and right side of the screen respectively. The tool panel consists of tables and text correction tool. The image panel consists of newspaper's images and the function panel contains the user management controls.

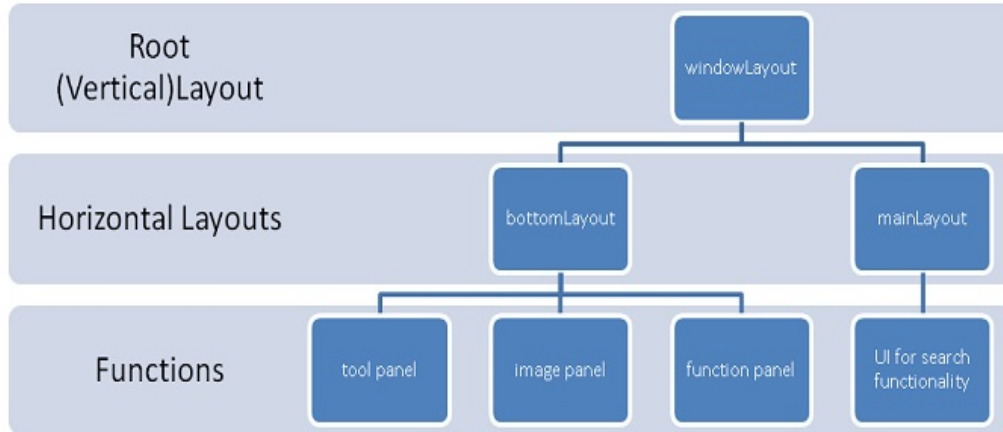


Figure 1: Interface Layout

Then the basic search functionality on the basis of issue names,date

range was incorporated. Once a issue is selected from the dropdown list and the search button is clicked, all the headlines of the selected issue appears in the table contained in the left pane. The images of the corresponding issue are displayed in the image panel. The headlines are in the form of links which when clicked shows the entire article in the table.

- In Phase 2, the actual implementation of collaborative text correction tool took place. Once the entire OCR text is retrieved in the rows of a table, it also gets highlighted in the image at the image panel. If "Edit Text" button is clicked, the OCR text becomes editable and a save button is displayed at the bottom of the left hand pane. A user can correct the text to match it with the text in the image. After the "Save" button is clicked, a check is performed against the database to determine which lines of text have been changed. Only those text lines that have been changed are saved as a new version.
- In Phase 3, user management functions, advance search options like phrase searching, enhanced UI features and other functionalities of the tool will be focussed.

4 Design Models

4.1 Database Design

The ER Diagram shown in Figure 2 describes the association between the tables present in the database of this project. All the entities are connected to a common entity called "Issue_Page_Article_Relator". It contains the primary keys of all other entities. The main entities are "Issue" that refers to the Newspaper, "Page" that denotes the pages in the newspaper and Article related entities called "Article_info", "Article_OCR_File", "Article_Service_File". "TextBlock" and "String" are used to fetch and group all the words in a single article in proper order, hence their attributes include the Horizontal and Vertical position of each textline. In addition to the above, a table was created to save the changes made in the OCR text and to save the user specified tags in the database. This was created due to the challenges faced as mentioned in section 6.

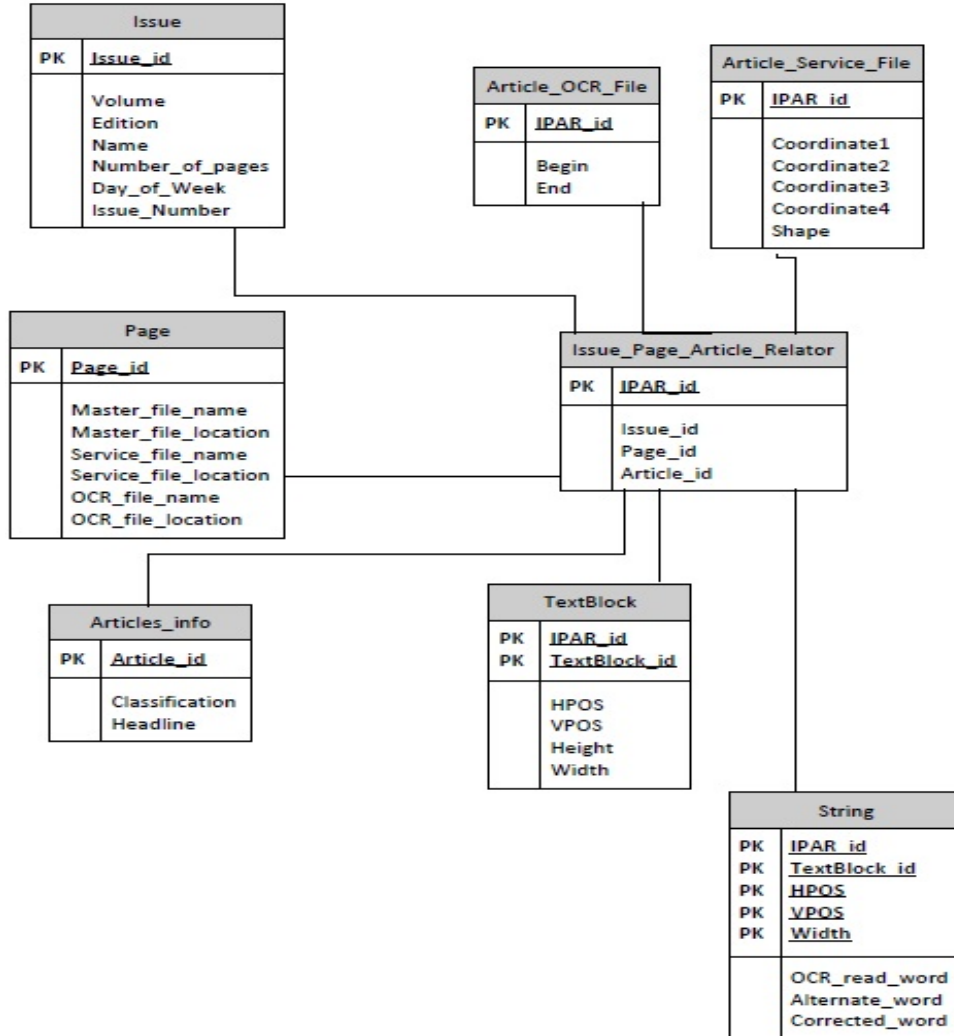


Figure 2: Database Design

Figure 3 shows the table created for storing OCR correction and tages. The “id” is bigserial Primary Key, “newtext” is the user corrected text, “tag” is the user specified tag corresponding to a given article and “ts” refers to the timestamp when the insert operation is executed.

Views are used to access the tables present in the database. A View is a virtual table containing fields from one or more real tables. They make



Figure 3: Experimental table

View Name	Description
vw_issu_page_files	retrieves the page file locations related to particular issue using the "date_of_issue" column
vw_issue_articles	accesses all the articles related to a particular issue using the "date_of_issue" column
vw_image_info	accesses the coordinates, shapes and related data for each article using the "ipar_id" column
vw_article_text	accesses entire text related to a particular article using the "ipar_id" column

Table 1: Views used by the web interface.

information retrieval fast and secure. Views used by the web interface are described in Table 1.

4.2 Architectural Design

The architecture used in this project is the Model-View-Controller (MVC) architecture. MVC is the software architecture pattern that separates the representation of information from user's interaction with it. It is used by applications that need the ability to maintain multiple views of the same data. The interaction between the three components is defined as:

- A controller sends commands to its associated view to change the view's presentation of the model. It can also send commands to the model

to update the model's state. It processes every request, prepares other parts of the system like model and view.

- A model handles data processing and connections to the database. It processes events sent by controller. After processing these events, it sends processed data to controller (thus, controller may reprocess it) or directly to view side allowing them to produce updated output.
- A view prepares the interface to show to the user. It requests from the model the information that it needs to generate an output representation to the user.

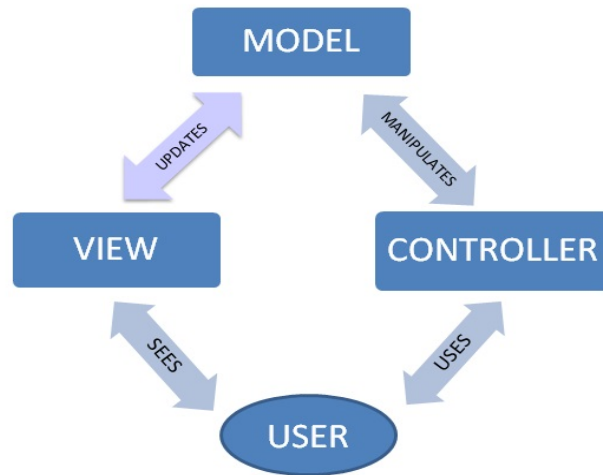


Figure 4: MVC Architecture

4.2.1 Implementation

In this project, the class `MainWebApp.java` is used as the controller which receives calls from the user and in turn calls the relevant screen (model or view) based on the input. `IssueDisplay.java` acts as a view. This class is the primary display page for an issue, shown one page at a time. It takes in the following parameters - list of articles in an issue and the page image to be displayed first. It then display the list of articles as hyperlinks in the left pane, along with the issue title and display the image in the right

pane. DatabaseHelper.java acts as a model contains any database objects that needs representation.

5 System Study

5.1 Hardware Specification

The hardware development environment comprised of the following:

- 32-bit Operating System
- 4GB RAM
- Intel(R) Core(TM) i3 CPU @ 2.13 GHz

5.2 Software Specification

The software development environment comprised of:

- Eclipse Juno 4.2 IDE: It is an Integrated Development Environment (IDE) comprising of a base workspace and an extensible plug-in system for customizing environment. It is used to develop applications primarily in Java and other languages by means of other plug-ins. Vaadin plug-in for eclipse was used to create the web interface of our project.
- Vaadin 6 (Java Framework) plugin for Eclipse: Vaadin is an AJAX web application development framework that enables developers to build high-quality user interfaces with Java, both on the server- and client-side. It provides a set of libraries of ready-to-use user interface components and a clean framework for creating our own components. The focus is on ease-of-use, re-usability, extensibility, and meeting the requirements of large enterprise applications. It is an open source web application for rich Internet applications. It has server side architecture which means that the majority of the logic runs on the server. Ajax technology is used at the browser side to ensure a rich and interactive user experience. On the client-side Vaadin is built on top of Google Web Toolkit (GWT).

- Apache Tomcat 7.0.35: It is an open source web server and servlet container developed by the Apache Software Foundation (ASF). It implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a pure Java HTTP web server environment for Java code to run in. It includes tools for configuration and management, but can also be configured by editing XML configuration files.
- Apache Ant 1.8.4: It is a software tool for automating software build processes. It is similar to "Make" but is implemented using the Java language, requires the Java platform, and is best suited to building Java projects. The most immediately noticeable difference between Ant and Make is that Ant uses XML to describe the build process and its dependencies, whereas Make uses Makefile format.
- PostgreSQL: It is an open source object-relational database management system (ORDBMS) available for many platforms including Linux, Microsoft Windows and Mac OS X. It implements the majority of the SQL:2008 standard, is ACID-compliant, is fully transactional (including all DDL statements), has extensible data types, operators, index methods, functions, aggregates, procedural languages, and has a large number of extensions written by third parties.
- Tortoise SVN 1.7: A Subversion client for Windows. It is a free software that helps programmers manage different versions of the source code for their programs. Tortoise SVN is a Subversion client, implemented as a Microsoft Windows shell extension. It is released under the GNU General Public License.
- Web Browser, Google Chrome or Mozilla Firefox

5.3 Setting up the code base

Following are the steps to get the application working:

1. Checkout the Bodhi Webapps code located here - <https://power.ldeo.columbia.edu/svn/Proj/NYPL/Bodhi/Code/WebApps> using Twiki username and password.
2. Verify that it has the following directory structure:

- build (directory where the classes of some components will be located, for now, this is empty)
 - config (This folder should have two files - log4j.properties and BodhiWebApplication.properties) Edit log4j.properties, change the value for “log4j.appender.A1.File” to a specific directory on the machine. No changes are required in BodhiWebApplication.properties
 - doc (This is the directory for any documents, including javadoc)
 - resources (If the application is going to use any external resources, they will be stored here)
 - src (Root folder for all the source files)
 - edu.columbia.ccls.dh.bodhi.controller (Folder for all the controllers, these will have the logic to direct the calls to specific components. It will have the code to instantiate the models and views corresponding to the action chosen by the user)
 - edu.columbia.ccls.dh.bodhi.model (Folder for all the model classes, typically any database objects that need representation)
 - edu.columbia.ccls.dh.bodhi.view (Folder for constructing the HTML views. Information needed for constructing will be passed to it by the constructor, typically either an instance of the model or specific values from the model)
 - edu.columbia.ccls.dh.bodhi.utils (Folder for all the utility classes and functions. Components in this folder will be used across the board, like database connections, special string operations, validations.)
 - WebContent (Root folder for all the Vaadin files as well as the web application classes)
3. Set up the project in Eclipse, just by choosing create new project from existing source and pointing it to the root folder where the entire folder is checked out.
 4. Once the project is created, make sure that the java path is set correctly (it might be different on different machines)
 5. Once the project is ready, open the deploy.xml, locate the entry , change the value of the attribute “location ”, point it to the Tomcat installation directory on the machine.

6. Make sure Apache Ant is installed on the machine (if not, download it from apache.org and make sure that the executable location is included in the PATH variable)
7. Verify that there are no errors, then open up a command prompt
8. To connect to the database on Tesla.ldeo.columbia.edu, a tunnel needs to be established. To do this, type
`ssh -Nn -L 5555:localhost:5432 username@tesla.ldeo.columbia.edu &`
Replace the username and password with the corresponding user's name and password.
9. Open another command prompt, change the directory to the root folder of the project and type `ant war`. This will compile the project, create a war file and deploy it to the Tomcat's "webapps" directory.
10. Start Tomcat (if it is not already running), go to bin and run `startup.bat` to start the server and `shutdown.bat` to shutdown the server.
11. Open up a web browser, go to <http://localhost:8080/bodhi/MainWebApp>, to see the page, Bodhi Issue Display created by the application.
12. To verify that the application is up and running correctly, open up the log file that was specified in step number 2

6 Implementation

6.1 Functions Implemented

1. List of Issues : Populated the combobox with the retrieved list of issues from the database. Illustrated in figure 5
2. Article Search : Retrieved articles based on a set of criteria like article topic, issue name, date range. Illustrated in figure 7
3. Headlines : Retrieved articles are the headlines in the form of links which when clicked returns the OCR text. Illustrated in figure 7
4. Images: Retrieved images of the selected articles in the middle pane.

5. OCR Text : Displayed the raw OCR article text of the user selected headline in the table. Illustrated in figure 8
6. OCR Correction : Raw OCR text can be edited after clicking the "Edit" button. Once the button is pressed, each row of the table becomes editable. Illustrated in figure 9
7. Tag Specification: Interface allows users to specify tags corresponding to the given article which then gets saved into the database. Illustrated in figure 10
8. Save corrected Text : Text corrected by the patron gets stored in the database as soon as the "Save" button is clicked. Illustrated in figure 11

6.2 User Interface

The User Interface is designed using Vaadin. Some of our project screenshots are shown in Appendix A.

6.3 Remote Debugging Tomcat using Eclipse

To debug our project, Eclipse Integrated debugger could not be simply used to debug as there was no an executable file. So in order to debug a Tomcat application through Eclipse, following steps need to be taken:

1. Open catalina.bat file which can be found in "apache-tomcat-7.0.35/bin/catalina.bat"
2. Insert two statements after "set CLASSPATH="


```
CATALINA_OPTS=
-Xdebug-Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n"
JPDA_OPTS="-agentlib:jdwp=transport=dt_socket,address=8000,server=y,suspend=n"
```
3. Save and close the file.
4. Run Tomcat from command prompt using command:


```
catalina.bat jpda start
```

 instead of using the normal startup.bat
5. Then in Eclipse create a debug configuration like below:
 - Give the project name.

- Give the connection type as Standard(Socket Attach)
- Put the host as localhost
- Specify port as 8000
- Give any name for configuration.

6. After setting the debug configuration, start debugger in Eclipse.

6.4 Challenges

There were several challenges faced during the project, few of them are listed below.

1. Firewall at server end: Since the database server was at an external location secured with firewall, there were problems in accessing the server at first.
2. Firewall at client end: After the access to the database was granted the problem arose due to Cyberoam layer. I could access the server from outside but not the institution due to restricted remote access in the presence of firewall. Finally I was allotted a port using which I could successfully create a secure tunnel.
3. Inaccessible Images: Images that were to be displayed in the web page were not available publicly as they were present only on the server. So hundreds of images of size 3.5GB were first securely copied (PSCP) to the client and then they were pushed to the cloud.
4. Incompatible Images: All the images that were available were of the format .jp2 which is JPEG 2000. It uses a wavelet compression algorithm instead of the common DCT compression method that is used with standard JPEG images. This format can be used to store either lossy or lossless compressed JPEG images and provides the user with a better quality and smaller file size than the traditional JPG image files. However, these images are not supported by the browser, hence they can not be opened in the browser.
5. Performance Issues : Due to the remote access to the server through tunneling and by using complex subquery to retrieve the OCR text, the server turned out to be slow. The time clocked for the server to

response for the query (shown below) was 16 minutes when using 3G network. These long delays and other problems made it difficult to work on real data.

An Example Query:

```
“ select linetext from vw_ipar_text1 where ipar_id IN (select ipar_id
from vw_issue_articles where headline =” + headline
```

Alternatively,

```
“select linetext from vw_ipar_text1 t1 INNER JOIN (select ipar_id from
vw_issue_articles where headline =`"+ headline + "`')t2 ON t1.ipar_id
= t2.ipar_id ”
```

The above queries fetches the entire article corresponding to the user selected headline. But both of them makes the retrieval process slow. So, the functionality to edit the raw text was built by setting OCR text in the table a priori and saving the edited data into the database. Figure 3 in Section 4 defines the table created to save the user corrected text and user specified tags.

7 Conclusion

The OCR [8] Text Correction System has not been fully implemented yet. There are several challenges as described above that require attention before proceeding further in terms of implementation. This system when functional will contain a repository of articles which would be clean and usable by many. The process of editing text will also help in building and growing of a community.

Once the system would be functional, it will certainly act as a rich source of information for historians, researchers, scholars and users in general.

8 Acknowledgement

This implementation project was a part of my M.Tech Scholarly work. I would like to express my gratitude to both my advisors Dr. Haimonti Dutta

and Manoj Pooleery for their constant support and guidance without which I would have never been able to complete it.

References

- [1] <http://ant.apache.org/bindownload.cgi>.
- [2] <https://vaadin.com/download>.
- [3] <http://tomcat.apache.org/download-70.cgi>.
- [4] <http://tortoisesvn.net/downloads.html>.
- [5] <http://www.eclipse.org/downloads/>.
- [6] <http://www.postgresql.org/download/>.
- [7] California Digital Newspaper Collection. <http://cdnc.ucr.edu/cdnc>, June 2009.
- [8] Line Eikvil. Optical character recognition. 1993.
- [9] Marko Grnroos. *Book of Vaadin: Vaadin 7 Preview edition*. Vaadin Ltd, 2013.
- [10] Allen Condit Kazem Taghva, Julie Borsack. *Evaluation of Model Based Retrieval Effectiveness with OCR Text*, volume 14. New York, US, 1996.
- [11] Allen Condit Kazem Taghva, Julie Borsack and Srinivas Erva. *Effects of Noisy Data on Text Retrieval*, volume 45. Las Vegas, US, 1993.
- [12] Manoj Pooleery. https://power.ldeo.columbia.edu/twiki/pub/Incubator/BodhiDBDesign/Final_ERD.pdf.

Appendices

A Screenshots

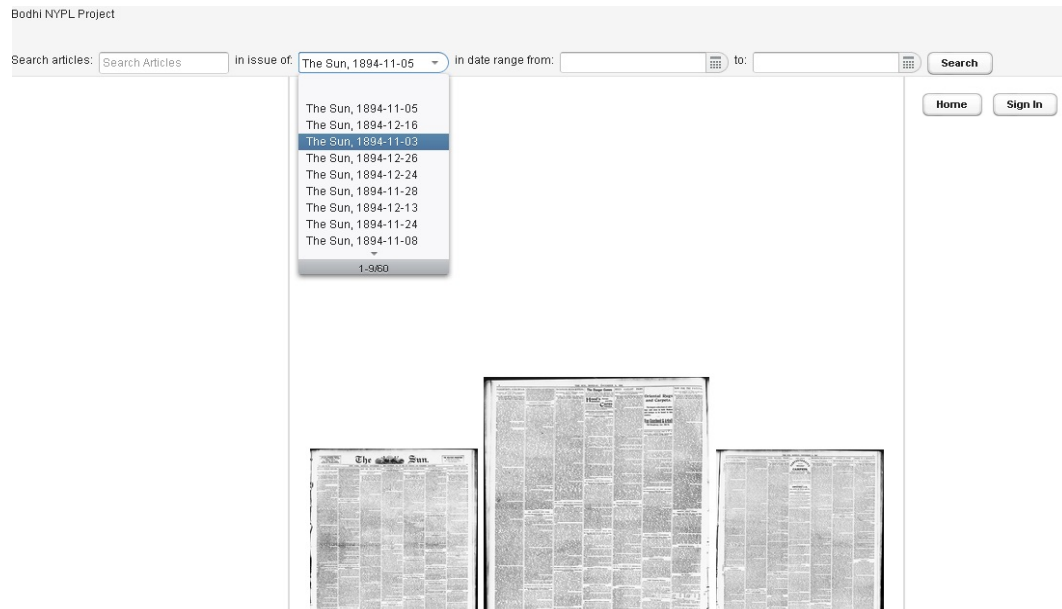


Figure 5: List of issues

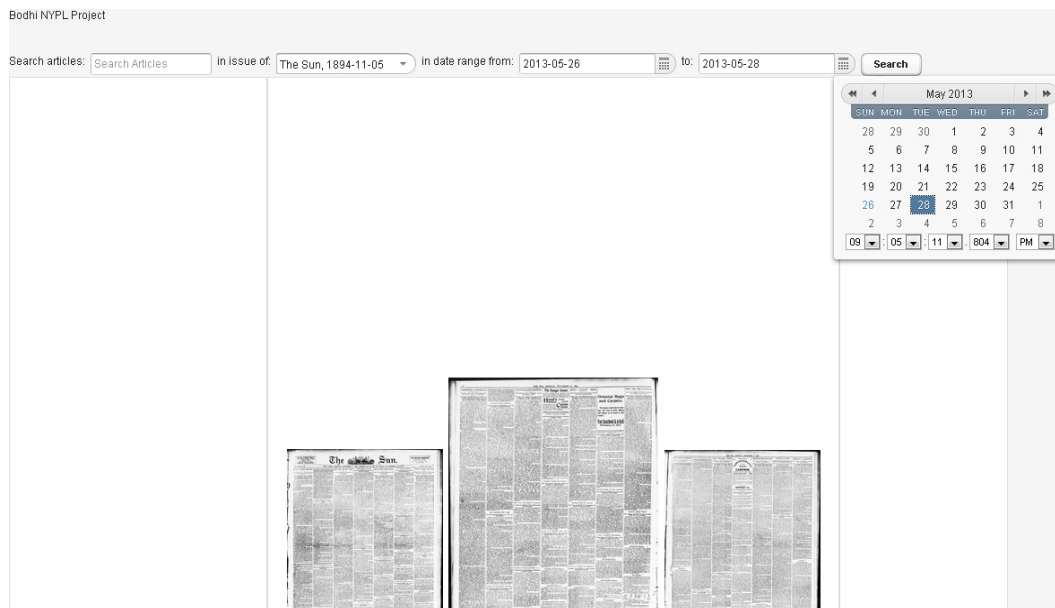


Figure 6: Datepicker



Figure 7: List of headlines in a issue

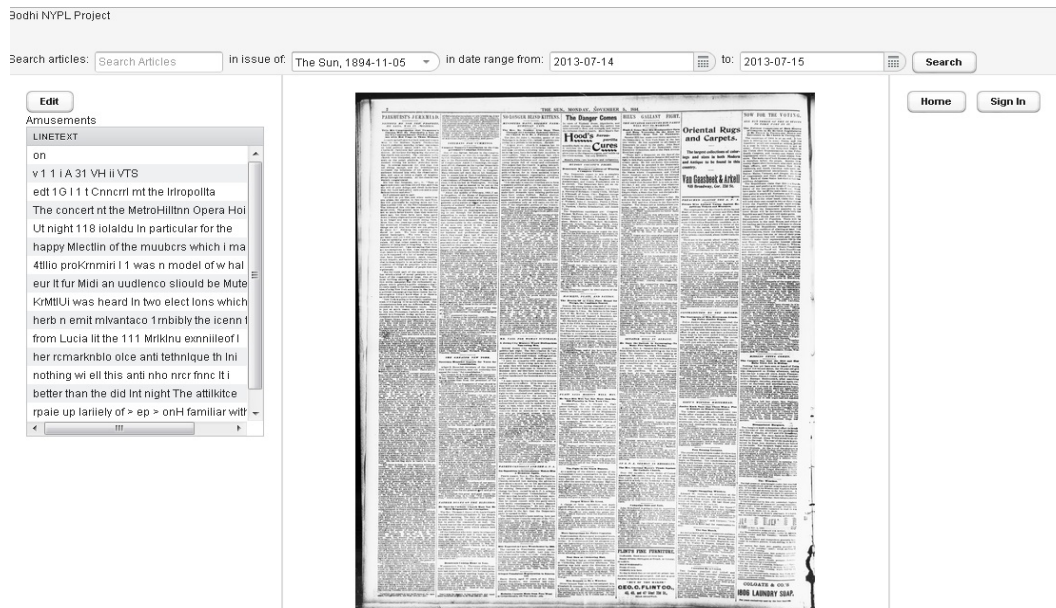


Figure 8: OCR Text of the headline 'Amusements'

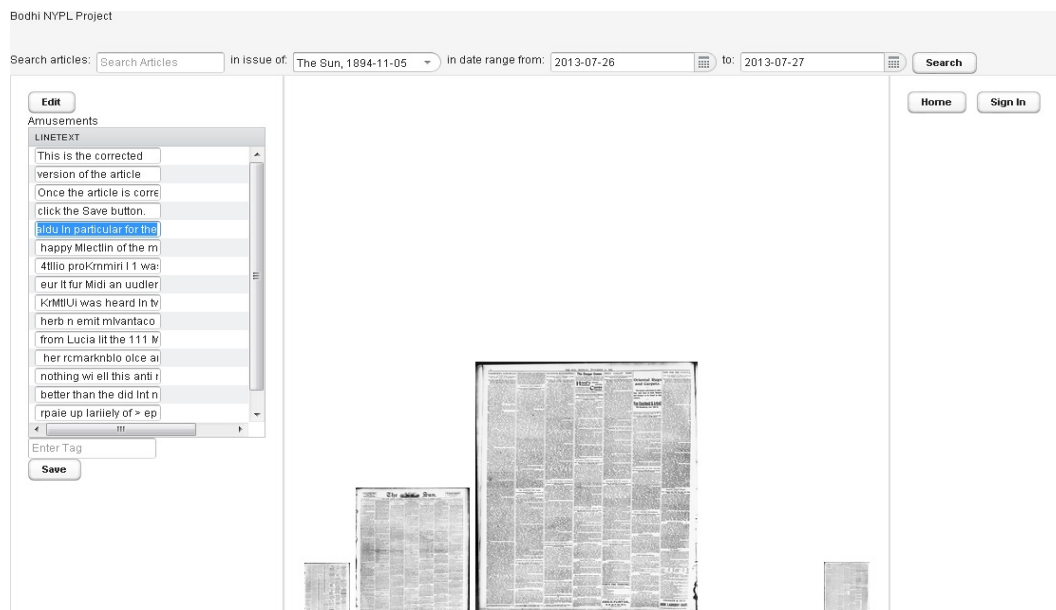


Figure 9: Editing the Raw Text

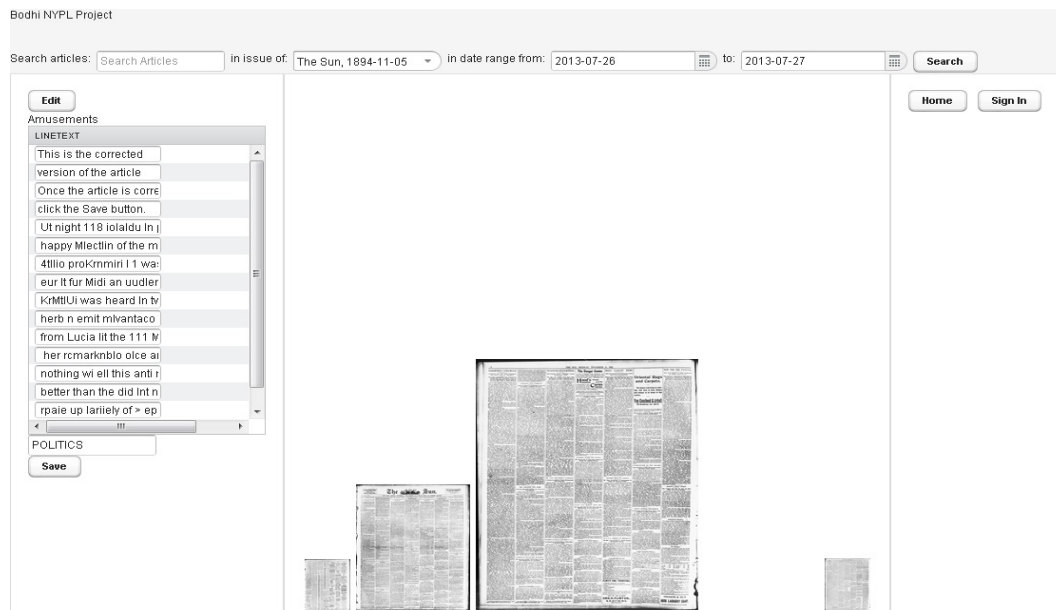


Figure 10: User Specified Tag

SQL Shell (psql)

```
nyp1test=> select * from usertext;
```

newtext	ts	id	tag
football	2013-07-26 14:51:33.297218	1	sports
elections	2013-07-26 14:52:15.684485	2	politics
movies	2013-07-26 17:29:02.635436	3	entertainment

(3 rows)

Figure 11: Snapshot of the table