IMPLEMENTATION OF BOOKS DIGITAL REPOSITORY SEARCHING USING HIBERNATE SEARCH SOFTWARE LIBRARY

Šnajderov Jovan*, Ivanović Dragan**
* Levi9 IT Services, Novi Sad, Serbia
** University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia
jovansna@gmail.com, dragan.ivanovic@uns.ac.rs

***Abstract*— A web application for searching through digital book repository is implemented. Server application indexes and persists book using Hibernate Search software library, exposing it's functionality through REST services. Client application developed using AngularJS, calls these services through web browser.**

I. INTRODUCTION

It was the month of March 1989, when British engineer Tim Berners-Lee finished his proposal project, for a system that would allow information flow between researchers at CERN physics department. Not long after that, Tim also developed first web browser, a tool for accessing web pages, written in format that would allow them to contain text, images, videos, and software components. This web browser made the communication with first internet server – possible. Date when the first internet web server at CERN became available over internet, 25th of December 1990, started a new era of global information.

Creating web, brought along new challenges. Amount of information that was enormous even before this global interconnecting, only kept growing with an unstoppable pace. Main challenge was extracting information from this new network. People searched for information on the web using links, mainly starting from hand created indexes from sites like Yahoo. These manually maintained and created lists covered the important topics efficiently, but subjectively. They were extremely expensive to create and maintain, and did not show any signs of improving or speeding up. Automatic search engines relied heavily on key words and most often presented too many bad matches. Until one day in 1996, two engineers from Stanford University presented a prototype of new search engine, with enormous scalability that relied heavily on text structure on web pages. This project started a revolution and changed the face of search, as we knew it.

Scale of this revolution is visible today, as well. Search is a key component of our digital lives (Google, Facebook, Amazon…). On every web page, in every application, we expect highly intelligent data search.

In this paper, we present a digital books repository implemented as modern web application, which in its core contains software tool that allows for advanced data search and low level data indexing using the Hibernate Search library.

Point of this paper is to present use of Hibernate Search Java library as well as to highlight advances in development of search tools, through a real world example. Quality of search and search results that we've got accustomed to, using modern search system, is now available for easier implementation in our own and specific applications.

II. BACKGROUND

Almost all applications require data persistence. If the information system would not persist data, all of the data would be lost on system shut down and with it, any practical use would be lost. When it comes to Java persistence, it usually relates to saving data into relational data bases using SQL. This process usually consists of large amount of repetitive boilerplate Java code. However, there is a solution that tackles this issue from Java code's perspective - object relational mapping.

A. Hibernate

Object relational mapping (ORM) is automatized and transparent persisting of objects into tables of relational databases, using metadata that describe mapping between these objects and the database [1; 2]. Hibernate is a complete ORM tool created as independent, non-commercial, open-source project from which the Java persistence specification came to be. This Java persistence API describes data access, data persistence, and data management between Java objects/classes and relational databases. Data extraction from relational databases should also allow end user to search for information. By that, we mean process of search of documents, search of information inside documents, and metadata about documents.

B. Lucene

Apache Lucene is very powerful and widespread tool for information search [3], but if we try to integrate it into existing software solutions, we quickly come across its flaws. Problems that Lucene faces are the following mismatches:

1. Structural mismatch – converting object domain into index in text form; dealing with connections between objects in index.
2. Synchronous mismatch – how to maintain database and index synchronized the entire time.
3. Retrieval mismatch – how to sustain smooth integration between domain models oriented methods for data retrieval and full text search.

In light of this mismatches, another project emerged, Hibernate Search.

C. *Hibernate Search*

Hibernate Search is Java library that represents integration of Hibernate ORM solution and Lucene [4; 5]. It is a project complementary to Hibernate ORM library, allowing full text search using queries directly over persistence domain model of the data. Technology that lies beneath Hibernate Search project relies completely on Apache Lucene. It hides complex usage of Lucene API, using simplified controls for advanced Lucene functionality, and allows us to index and retrieve persistence domain data from Hibernate, with minimal effort. The Hibernate Search software library has been previously used in various projects [6; 7].

III. SYSTEM SPECIFICATION

Primary use cases necessary to implement in the digital library were:

1. Adding a book – adding book metadata and the book content itself.
2. Updating a book – updating existing data and metadata related for any and every book.
3. Book search – advanced book search using specter of different search parameters (metadata or the contents themselves).
4. Reviewing a book – inspecting data of existing books (look at the metadata and possibility of book download).
5. User subsystem – access to repository user records and tracking access permissions to different parts of the system.

IV. SYSTEM ARCHITECTURE AND IMPLEMENTATION

System architecture is shown in Figure 1. System consists of server application and client application. Server application exposes services to handle books, implements business logic, enables user manipulation, persists data and it indexes books. Data is persisted in PostgreSQL database that we access from inside of the server application over the ORM tool (Hibernate). We index books (metadata and e-book contents) using Hibernate Search, and we keep its index on a server's file system. Main server application is available over application server that exposes interaction with available services over http protocol.

Client application uses these exposed server side REST services to allow users to add a new book, to search for a book, to update book metadata, to create user account, etc. Core of the project, simplicity of adding advanced search capabilities will be presented on the main data model, the EBook entity shown on Listing 1.

To enable full text capabilities over our entities, we first need to add couple of annotations. First, we annotate that the entity should be indexed using Hibernate Search @Indexed annotation. Second, we define entity identifier that Hibernate Search saves in its index for every entity. For this purpose we set @Id annotation on identification attribute of our class and our primary key in eBook database table.

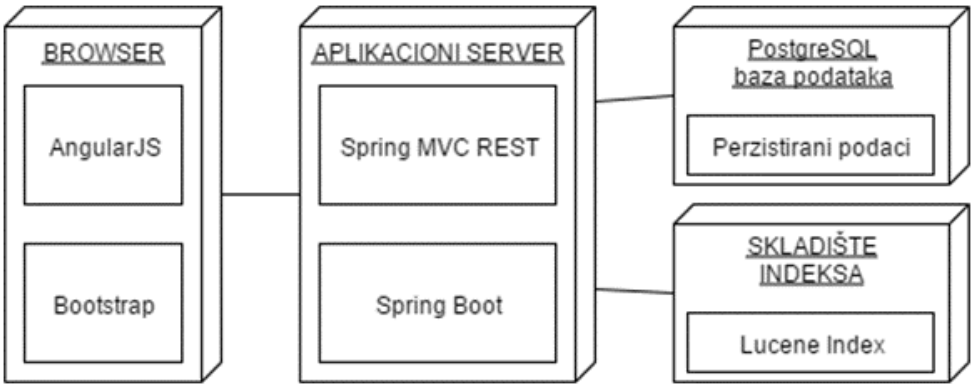

Figure 1 Deployment diagram

Next step is to determine which fields we want be able to search in our digital repository. Annotation that we use for this is @Field. Annotation parameters that we can set are as follows. Parameter index = Index.YES assures that the text will be indexed. Parameter analyze = Analyze.YES assures that the text will be analyzed using default Lucene analyzer. Third parameter store = Store.YES assures that the data themselves won't be saved in index. Saving attribute data inside index does not have anything to do with the ability to search them. Advantage that this parameter enables is ability to retrieve attributes value over projections. When we do not use projections, Hibernate Search uses Lucene queries to find entity identifiers in the database, so it could use them to retrieve managed objects from the database. Using projections, we could avoid this trip to the database and back, but not without cost: this way we would get array of objects and not managed object that we get with regular query. One option allows better performance, while the other enables ease in development.

```
@Entity
@AnalyzerDef(name = "customanalyzer", tokenizer =
@TokenizerDef(factory = StandardTokenizerFactory.class),
filters = {
    @TokenFilterDef(factory = LowerCaseFilterFactory.class),
    @TokenFilterDef(factory =
SnowballPorterFilterFactory.class, params = {@Parameter(name
= "language", value = "English")})})
@Indexed
@Table(name = "EBOOK")
@Cache(usage =
CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
public class Ebook implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Field(index = Index.YES, analyze = Analyze.YES, store =
Store.NO)
    @Column(name = "title")
    @Analyzer(definition = "customanalyzer")
    private String title;

    @Field(index = Index.YES, analyze = Analyze.YES, store =
Store.NO)
    @NotNull
    @Column(name = "creator", nullable = false)
    private String creator;

    @Field(index = Index.YES, analyze = Analyze.YES, store =
Store.NO)
    @Column(name = "description")
    @Analyzer(definition = "customanalyzer")
    private String description;

    @Type(type =
"org.jadira.usertype.dateandtime.joda.PersistentLocalDate")
    @JsonSerialize(using = CustomLocalDateSerializer.class)
    @JsonDeserialize(using =
ISO8601LocalDateDeserializer.class)
    @Column(name = "date", nullable = false)
    private LocalDate date;

    @Field(analyze = Analyze.YES, store = Store.YES)
    @TikaBridge
    @Analyzer(definition = "customanalyzer")
    private String content;
```

Listing 1. The Ebook entity

Advanced search options over our entities are achieved using more advanced annotation parameters. Let us assume that we indexed the book entity with the title "Hibernate Searching" and that we want to have this book as a matched result even if we search for key words "search", "searches", "searched" or "searching". To achieve this using Lucene; we need to set analyzer class just like in Listing 1. This analyzer sets word stemming while indexing and during search process. In our class, we use StandardTokenizerFactory that splits the words at punctuation marks and dashes, but maintains the format of the email addresses it comes across, as does with the internet domains. On this analyzer we chained two filters. First one converts all the letters to lower case (LowerCaseFilterFactory). Second filter adds specific grammar rules and syntax for the English language.

@TikaBridge annotation uses Apache Tika tool to allow for extracting text and metadata of the passed documents. This annotation uses attributes with type String, URI, byte[] or java.sql.Blob. Using String typed attributes, this tool recognizes values of attributes as file locations and tries to open them to parse the document saved at that location. During the eBook upload, we actually save the book on the server's file system, and repository only grabs a hold of the file location address inside the String content attribute. After persisting the eBook object, Hibernate Search reads location address from the content attribute, finds the book on file system, parses document contents, indexes text contents and updates the index for that particular book. We only have to take care that the right book contents path is present in the content attribute. Figure 2 depicts the form for input book metadata and uploading a digital file which represents the certain book.

Figure 3 depicts the look of the digital repository search page. Using this page, we call the method:

```
public List<Ebook> search(EbookSearchObject ebookSearchObject)
```

We pass the search object to this method. This method takes this object and parses its list of search criteria. Using these criteria, we get Lucene query suitable for searching Lucene index:

```
luceneQuery = MultiFieldQueryParser.parse(paramValuesArray,
paramNamesArray, flagsArray, new StopAnalyzer());
```

Given query, we wrap inside Hibernate query:

```
org.hibernate.search.jpa.FullTextQuery jpaQuery =
fullTextEntityManager.createFullTextQuery(luceneQuery,Ebook.
class);
```

Execution of previous jpaQuery retrieves the list of results.

We just showed how, using just a couple of methods, we raised the whole complexity of the entire advanced search on the higher level of abstraction. With this, we saved ourselves from writing great amount of boilerplate code, leaving that part of the job, to the specialized library – Hibernate Search.

V. CONCLUSION

Quality data search is necessary for managing almost endless ocean of information that is out there. To average user of the global network, it would be beyond imagination to retrieve any usable valued information if it is not available through simple search and retrieval system. This complete dependency on our search tool shows just how clear goal of further advances in this field is. Every meaningful advance in this field of computer science represents a gold mine of its own kind. Gold mine that could almost the entire world population is waiting to use regularly. This claim is backed by the value of the company behind the most influential Internet search engine - Google.

Search technology presented in this paper, using Hibernate Search library, represents simple all-round solution for quick and efficient enabling of advanced search capabilities to any system. Tools like this software library, open the door for new researches, and create new possibilities for engineers looking to push the envelope in the field of search.

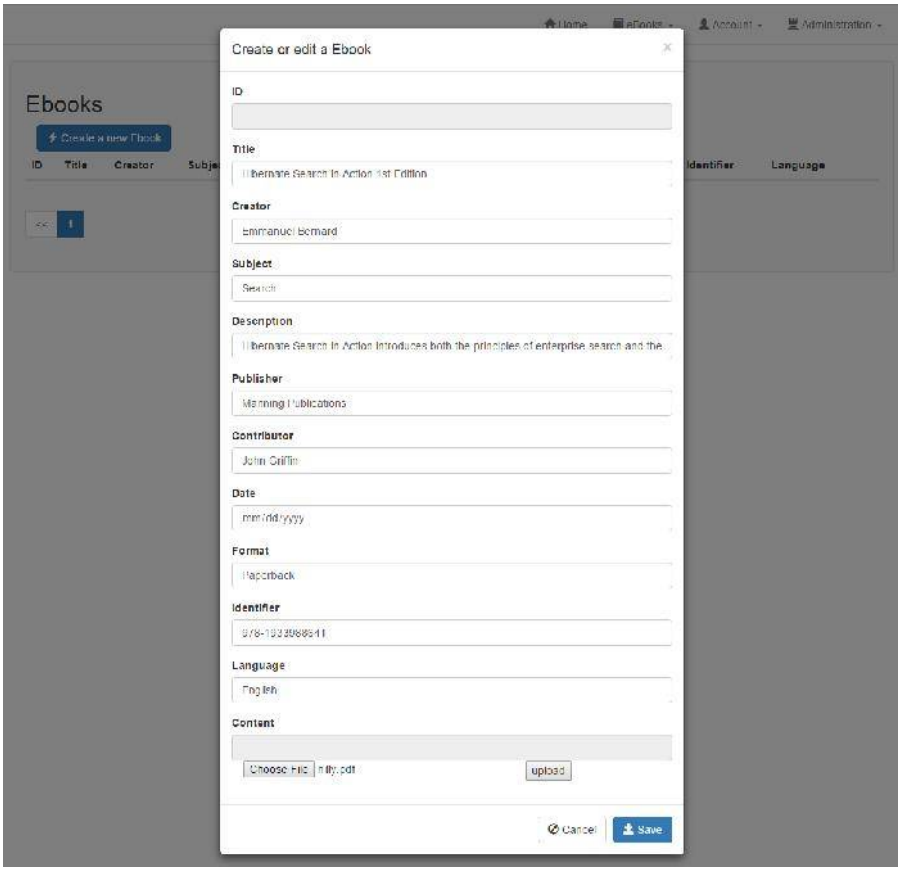

Figure 2 Dialog for input new book

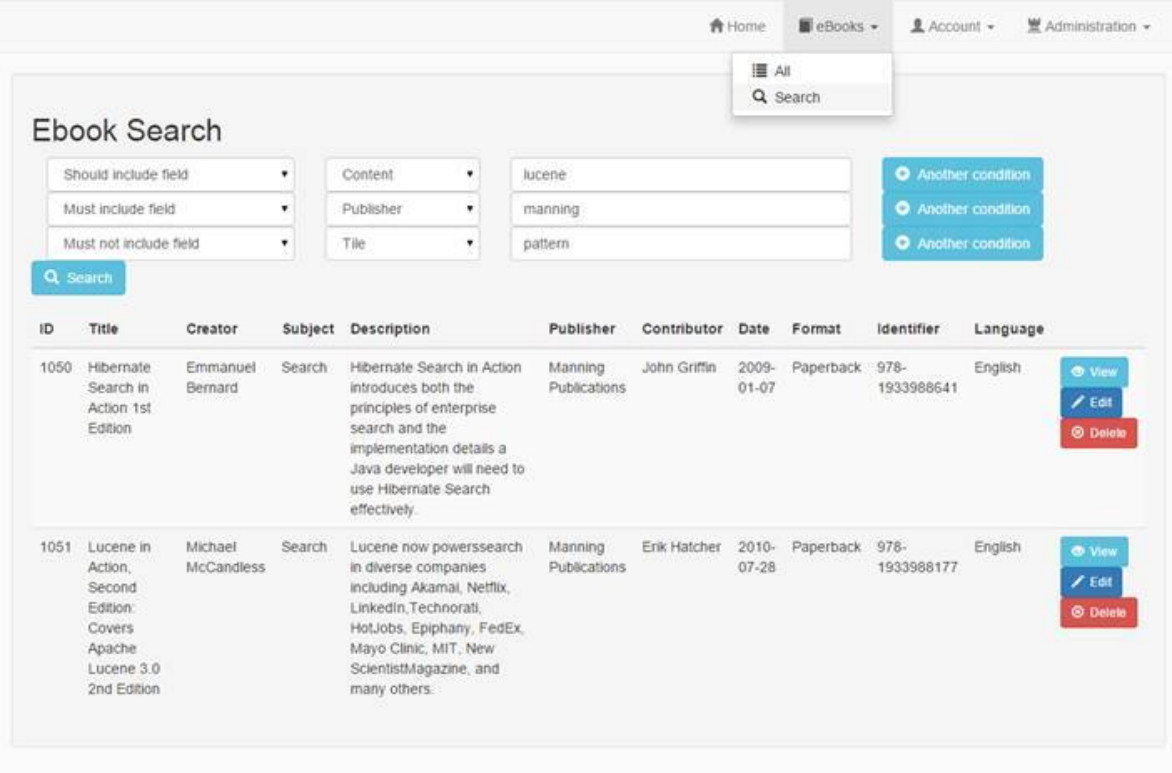

Figure 3 The repository search page

REFERENCES

[1] C. Bauer, G. King, G. Gregory, *Java Persistence, 2nd edition*. Manning Publication, 31. oktobar 2015.; ISBN 978-1617290459

[2] C. Bauer, G. King. *Hibernate in Action*. Manning Publication, 28. avgust 2004.; ISBN 978-1932394153

[3] E. Hatcher, O. Gospodnetić, M. McCandless, Lucene in Action, 2nd edition. Manning Publication, 28. jul 2010.; ISBN 978-1933988177

[4] E. Bernard, J. Griffin, *Hibernate Search in Action, 1st edition*, Manning Publication, 7. januar 2009.; ISBN 978-1933988641

[5] S. Perkins, *Hibernate Search by Example*, Packt *Publishing, 26. mart 2013.; ISBN 978-1849519205*

[6] K.M. Anderson, A., Schram, *Design and implementation of a data analytics infrastructure in support of crisis informatics research (NIER track)*. In *Proceedings of the 33rd International Conference on Software Engineering* (pp. 844-847). ACM. 2011.

[7] Y.A.N.G., Qi, L.I.N., Zhen-can, H.U.A.N.G., Fan, X.I., Jian-qing, *Database Full-Text Retrieval System Based on Hibernate Search*. *Computer Engineering*, *4*, p.029, 2010.