

# CS5500 - PHASE 4

# REPORT

---

Paulomi Mahidharia

Abhinav Mourya

Aswin Gopalan


Rushikesh Badami

17th April, 2017

## Description of system functionality

The following are the system functionalities of Author Retrieval System:

1. **User Login:** User can login into the system using a username and password.
2. **User Registration:** User can register into the system using a username and password.
3. **Search:** User can search for authors based on the following criterias:
  - Search based on publication information: This is searching based on information associated to the publications by authors. It includes search based on one or more of the following criteria:
    - Minimum number of publications
    - Published/not published in one or more conferences
    - Publications done before or after specific year, or in a year range
    - Title or keyword of the publication.
  - Search based on service information: This is searching based on information associated to the service by authors. It includes search based on one or more of the following criteria:
    - Served/Not served in one or more conferences
    - Service offered before or after specific year, or in a year range
    - Position of the author during the service
  - Search based on author information: This is searching based on an author's name. It includes search based on the following criteria:
    - Matching author name
4. **View Author Profile:** User can view the following information on an author's profile page:
  - Author's name
  - Author's affiliated university
  - Author's webpage URL
  - List of publications by the author
  - List of committees served on by the author
  - User can perform the following important functionalities on author's profile page:
    - **Go to an author's webpage** which is a webview embedded in new window.
    - Shortlist an author.
5. **View Shortlisted Authors:** User can shortlist authors that he/she finds interesting from an author's profile page. At any point user can view the shortlisted authors, remove an author from the list, or select an author from the list to view his profile.

- 
6. **Export Author List:** User can export the list of authors obtained as search result or the list of shortlisted authors in PDF format.
  7. **Sort Authors:** On the result page, user can see the list of authors easily with the help of pagination. At any time user can sort authors on a single page in ascending and descending format.

## Enhancements


The following enhancements/changes are incorporated in the system as part of Phase 4:

### BACK END ENHANCEMENTS:

1. **Additional Data Sources:** We incorporated data from CSRankings.org such as author's affiliated university information and author's homepage URL.
2. **Database enhancement:** Refactored database tables for faster query execution which includes restructuring tables, creating indexes, creating mapping tables, and applying normalization. This has resulted in approximately 70% increase in the query execution time.
3. **Design patterns:** Implemented Singleton design pattern for Database connection so that it is not created repeatedly. Also, implemented abstract factory design pattern to instantiate objects for different parsers associated with different data sources.

### FRONT END ENHANCEMENTS:

1. **Progress Indicator:** For the action events that query the Database and will require minimal time to extract information, we are showing a progress indicator in the UI to reflect this action. This awares the user about data being fetched and an action being ran in the background. Internally we are creating threads that execute the query in the background and track the progress.
2. **Stack-based navigation:** Stack-based navigation functionality across multiple scenes has been implemented as an enhancement of Phase 4. The lifecycle of the Stack is from user login to user logout. Every time a scene comes to the foreground, the previous scene is pushed to the stack and the current scene is maintained in a global variable of Stack util class. On click of "Back" button at any scene, the scene at the TOP of the stack is popped and displayed. On click of "Logout" button, the stack is flushed.
3. **Pagination:** Since the result page may have to display a lot of authors based on the user's criteria selection, it is difficult to scroll through or even look at so many records. To



overcome this, we implemented pagination. This lets you look at only few records at a time and sort those records.

4. **Shortlist author:** Favourite authors can now be shortlisted from their respective profile page. List of shortlisted authors for a particular user is saved and can be looked at when he/she logs in next time. At a given time, a shortlisted author can be removed or his/her profile can be looked at again.
5. **Author's homepage in a Webview:** From a author's profile page, if the user clicks at the author's homepage URL, a new window is opened and a webview is loaded that displays the author's homepage.
6. **Export results:** Exporting important information that the user might want to look at in future was taken care of in Phase 4. The user can now export the Author Results and Shortlisted Authors with a button click.

## Summary of changes since phase III

In addition to the above mentioned enhancements, the following changes have also been incorporated in the system during Phase 4:

1. **Bug fixes:** In response to the issues raised by the testers as part of Homework 4, we fixed all major issues(bugs) and various minor issues based on their priority. The issues not fixed were documented and added to the backlog(most of these issues were either related to the user experience, design decisions or out of the scope of current system's functionality).
2. **Validations and enhanced user experience:** Missing validations were added and incorrect validations were fixed during Phase 4. Examples of this are incorrect REGEX matching, checking for null objects, popping user friendly alerts for exceptions, etc.
3. **Query optimization:** We optimized existing queries for faster retrieval of author's information. For example, a complex query was broken down into simpler ones and the results were aggregated in the application. This also improved the readability and maintenance of the code.

## Refactoring, Design patterns, and Testing

### REFACTORING:

- As part of refactoring, we aimed at removing duplicated code, make as many constants as possible, and created utility classes.
- We also categorized and packaged constants and utility classes for the same.

- Streamlining migration process and documentation was a major part of Phase 4.
- Also, we attempted to reduce the complexity and enhanced structuring of the existing code.
- We also refactored the database for faster query execution as mentioned in the enhancement section

## DESIGN PATTERNS:

- Implemented singleton design pattern for instantiating the Database connection only once in the system's lifecycle. This is because it is a Desktop application and only one user is going to use it locally at a time. This makes Database operations efficient.
- Implemented Abstract Factory design pattern to instantiate different types of parser objects each corresponding with a different data source. The parser types in our application are as follows:
  - **DBLPParser**: Parser to parse entire DBLP data
  - **CommitteeMemberParser**: Parser to parse committee members given in text files by Prof. Tip.
  - **CSRankingParser**: Parser to parse author affiliation information from CSRankings.org
  - **AuthorHomepageParser**: Parser to parse author homepage URLs from CSRankings.org.

The skeleton of the **AbstractParserFactory** is as follows:

```
public abstract class AbstractParserFactory {

    public abstract AuthorHomepagesParser makeAuthorHomePagesParser(String path);
    public abstract CommitteeMembersParser makeCommitteeParser(String path);
    public abstract CsRankingsParser makeCsRankingsParser(String path);
    public abstract Parser makeDblpParser(String path);

}
```

All the parsers are instantiated in the **StandardParserFactory** that extends the **AbstractParserFactory**:

```
public class StandardParserFactory extends AbstractParserFactory {

    @Override
    public AuthorHomepagesParser makeAuthorHomePagesParser(String path) {

        return new AuthorHomepagesParser(path);

    }

}
```

```

@Override
public CommitteeMembersParser makeCommitteeParser(String path) {

    return new CommitteeMembersParser(path);
}

@Override
public CsRankingsParser makeCsRankingsParser(String path) {

    return new CsRankingsParser(path);
}

@Override
public Parser makeDblpParser(String path) {

    return new Parser(path);
}
}

```

Individual parser classes extend from the base class **ParserBase** and implement their own version of `executeParser()`:

```

public abstract class ParserBase {

    public abstract void executeParser();
}

```

## TESTING:

- UNIT TESTING
  - Failing test cases(mostly due to stale data) were fixed.
  - Additional test cases were added for improving **conditional** and **line** coverage.
  - Unit tests for the parser were added as part of the Phase 4.
  - Packaged tests into test suites corresponding to the packages in the main java package.
- FUNCTIONAL TESTING
  - For each feature, every team member who did not work on that feature, performed testing to verify the functionality of the feature.
- SANITY TESTING
  - At the end of each Sprint, sanity testing was performed to ensure overall expected execution of the system.
- END-TO-END TESTING
  - At the end of each Phase, entire system stack involving Front-end(includes UI), Query engine and Back-end(includes DB) was tested to verify the correct information flow.