Unit - 8 Assignment

Building an application to organise my information resources

The objective of this project is to design and develop a digital "Interest Book" application, which serves as a personal repository for collecting, organizing, and retrieving web links related to academic and professional interests. This tool aims to enhance information management by allowing users to categorize links using tags, facilitating quick access to relevant resources based on topics or keywords (Cormen et al., 2009).

The application will be developed in PHP and is designed to run locally on a personal computer, either through a browser (if using a GUI) or terminal (for command-line interaction). Users can add, view, search, and delete records using a simple menu-based interface or web form. Each entry consists of a title, a valid URL, between 1 and 5 descriptive tags (e.g., AI, privacy), and a timestamp indicating when the record was created and accessed.

The development of such an application is grounded in the principles of data structures and algorithms, which are fundamental in computer science (Cormen et al., 2009). By utilizing data structures like arrays and hash tables, the application can efficiently store and retrieve links. The use of algorithms for insertion, deletion, and searching ensures that the application is both functional and efficient (Knuth, 1997).

Furthermore, the integration of tags, which can later be translated into labels, provides a foundation for future enhancements, such as incorporating artificial intelligence (AI) models to improve information retrieval. AI models can be trained on these labels to enhance the search functionality, making it more intuitive and efficient (Russell & Norvig, 2010).

This project aligns with the broader goals of enhancing personal productivity and information management, which are critical in both academic and professional contexts. By developing a personalized digital repository, individuals can better manage the vast amount of information they encounter, thereby improving their ability to access and utilize relevant resources effectively.

References:

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 3: Sorting and Searching* (2nd ed.). Addison-Wesley.
- Russell, S. J., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.

Installation and usage instructions

1. Navigate to the application's root folder (e.g., `InterestBook`).

2. For command-line use: Run `php start.php` in the terminal.

3. For web use: Start a local server with `php -S localhost:8000`, then open `http://localhost:8000` in a browser.

4. Interact via the menu (terminal) or form (web) to manage records.

Data Structure

Each record in the Interest Book is represented as an associative array in PHP (Cormen et al ., 2009). This array contains four fields: a string for the title of the resource, a string for the URI, an array of tags (limited to 1–5 string values), and a timestamp string representing the date and time the entry was created and accessed. All records are stored in a main array, allowing efficient iteration and manipulation. This design supports common operations such as adding, searching, deleting, and sorting based on timestamp or tags.

Name	Type	Description
title	String	The title of the resource found. Could be book title, document title or name of a specific website
uri	String	The unique reference identifier. This could be a document location, web address, or file location.
tags	Array	An array of strings that helps identify the information in the document or resource. It could be, not limited by, category, subjects or theme based.
created_at	String (ISO 8601)	When was the record created
accessed_at	String (ISO 8601)	When was the record access, views, read.

PHP implementation

```
$record = [
    "title" => "Essex University Unit 8 - Assignment",
    "uri" => "https://www.my-course.co.uk/mod/assign/view.php?id=1144419",
    "tags" => ["datastructures", "programming", "algorithms", "php", "data architecture"],
    "created_at" => "2025-03-22T15:45:00",
    "accessed_at" => "2025-03-22T15:45:00"
];

$dataset = [] //ex. initialised globally
```

Pseudocode

- **Add record**

```
function addRecord($title, $uri, $tags) {
    if (count($tags) > 5 || count($tags) < 1 {
        return "Error: too many tags. Must be between 1 and 5";
    }

    $record = [
        "title" => $title,
        "uri" => $uri,
        "tags" => $tags,
        "created_at" => time(),
        "accessed_at" => time()
    ];

    // $dataset[] = $record //Add record to our dataset
}
```

- **Search by single hashtag**

```
function searchByTag($tag) {
    $results = [];
    foreach ($dataset as $record) {
        $found = false;
        foreach ($record["tags"] as $currentTag) {
            if ($currentTag === $tag) {
                $found = true;
                break; // Stop searching
            }
        }
        if ($found) {
            $results[] = $record;
        }
    }
    return $results;
}
```

- **Delete record by URI**

```
function deleteByURI($uri) {
    foreach ($dataset as $index => $record) {
        if ($record["uri"] == $uri) {
            unset($dataset[$index]);
            return "Record deleted";
        }
    }

    return "Record not found";
}
```

- **Sort by created at (Newest first)**

```
function sortByCreationDateDesc($dataset) {
    $length = count($dataset);
    for ($i = 0; $i < $length - 1; $i++) {
        for ($j = 0; $j < $length - $i - 1; $j++) {
            // Convert to timestamps
            $time1 = strtotime($dataset[$j]["created_at"]);
            $time2 = strtotime($dataset[$j + 1]["created_at"]);
            if ($time1 < $time2) {
                $temp = $dataset[$j];
                $dataset[$j] = $dataset[$j + 1];
                $dataset[$j + 1] = $temp;
            }
        }
    }
    return $dataset;
}
```

- **Search by multiple-tags (one match)**

```php
function searchByTags($tags) {
    $results = [];

    foreach ($dataset as $record) {
        foreach ($tags as $tag) {
            $found = false;
            foreach ($record["tags"] as $currentTag) {
                if ($currentTag === $tag) {
                    $found = true;
                    break; // Stop by one match
                }
            }
            if ($found) {
                $results[] = $record;
                break; // Stop searching
            }
        }
    }

    return $results;
}
```

- **Search by multiple tags (all match)**

```php
function searchByAllTags($tags) {
    $results = [];

    foreach ($dataset as $record) {
        $allTagsMatch = true; // Start as true

        foreach ($tags as $tag) {
            $tagFound = false; // Check each tag
            foreach ($record["tags"] as $currentTag) {
                if ($currentTag === $tag) {
                    $tagFound = true;
                    break; // Tag found, stop searching for this tag
                }
            }
            if (!$tagFound) {
                $allTagsMatch = false; // If one tag is not found
                break; // Stop
            }
        }

        if ($allTagsMatch) {
            $results[] = $record; // Add to our result
        }
    }
```

```
    return $results;
}
```

- **Edit existing record (by using URI as key)**

```
function editRecord($uri, $newTitle, $newTags) {
    foreach ($dataset as &$record) { // & direct modification of the record
(pointer)
        if ($record["uri"] === $uri) {
            if ($newTitle !== "") {
                $record["title"] = $newTitle;
            }
            if ($newTags !== [] && count($newTags) <= 5 && count($newTags) >=
1) {
                $record["tags"] = $newTags;
            }
            return "Record updated";
        }
    }

    return "Record not found";
}
```

- **Search by keyword in title or tags**

```
function searchByKeyword($keyword) {
    $results = [];

    foreach ($dataset as $record) {
        $keywordFound = false;

        // Check if keyword is in title (manual string search)
        $title = $record["title"];
        $titleLength = strlen($title);
        $keywordLength = strlen($keyword);

        for ($i = 0; $i <= $titleLength - $keywordLength; $i++) {
            $match = true;
            for ($j = 0; $j < $keywordLength; $j++) {
                if ($title[$i + $j] !== $keyword[$j]) {
                    $match = false;
                    break;
                }
            }
            if ($match) {
                $keywordFound = true;
                break;
            }
        }

        // Check if keyword is in tags (manual array search)
        if (!$keywordFound) {
            foreach ($record["tags"] as $tag) {
                $tagLength = strlen($tag);
                if ($tagLength >= $keywordLength) {
                    for ($i = 0; $i <= $tagLength - $keywordLength; $i++) {
                        $match = true;
                        for ($j = 0; $j < $keywordLength; $j++) {
                            if ($tag[$i + $j] !== $keyword[$j]) {
                                $match = false;
                                break;
                            }
                        }
                        if ($match) {
                            $keywordFound = true;
                            break 3; // Exit all loops once found
                        }
                    }
                }
            }
        }

        if ($keywordFound) {
            $results[] = $record;
        }
    }
```

```
    return $results;
}
```

Testing and Validation

To provide a comprehensive overview of the testing process, it is essential to detail the methodologies employed to validate the functionality of each function under various conditions. The testing regimen was designed to ensure that the core data structures and algorithms operate as intended, both under normal operating conditions and at boundary limits.

Testing normal conditions (happy flow)

Each function was tested with representative input data to verify that it performed its intended operations correctly. For example, the `addRecord` function was tested by adding multiple records with valid titles, URIs, and tags. The results confirmed that the records were successfully added to the dataset, and that the data structure maintained its integrity.

Testing under boundary conditions

Boundary conditions were thoroughly tested to ensure the robustness of the application. These included:

- **Tag Limit Enforcement:** The `addRecord` function was tested with more than five tags to verify that it correctly returned an error message, thereby enforcing the tag limit constraint.

- **URI Validation:** Tests were conducted to check whether URIs already existed and to handle cases where URIs were empty or invalid. This ensured that the application could distinguish between valid and invalid input.

- **Empty or Invalid Fields:** The application was tested with missing or empty values (e.g., title or URI) to confirm that it could handle such scenarios gracefully and return appropriate feedback.

Checkpoints

1. Record addition increases `$dataset` size.
2. Search returns the correct subset of records.
3. Deletion reduces `$dataset` size.
4. Sorting orders by `created_at` descending.

Test Results

- `addRecord("Test", "http://test.com", ["tag1"])`: Added with timestamps.
- `searchByTag("AI")`: Returned records with "AI".
- `sortByCreationDateDesc()`: Newest records first.

Validation of Input

Special attention was given to input validation to prevent potential inconsistencies or errors in the dataset. This included:

- **Data Type Validation:** Ensuring that input values matched the expected types (e.g., strings for titles and URIs, arrays for tags).
- **Data Format Validation:** Verifying that input adhered to proper formats (e.g., well-formed URIs).

Results and Conclusion

The results of these comprehensive tests confirm that the core data structure and its associated algorithms behave as expected. The application successfully handles both standard and edge case scenarios, thereby ensuring robustness and reliability. The thorough validation of input data further enhances the application's ability to maintain integrity and prevent operational errors. Code and results are stored in my GitHub repository: [https://github.com/va-angelier/va-angelier.github.io].

Test plan/ Test script

Test case	Function	Input	Expected outcome
Add valid record with 3 tags	addRecord()	Title, valid URI, 3 tags	Record is added successfully with timestamps
Add record with more than 5 tags	addRecord()	Title, URI, 6 tags	Error: "too many tags"
Add record with empty title	addRecord()	Empty title, URI, tags	Error or validation warning (if enforced)
Search by existing tag	searchByTag()	"AI"	Returns all records containing AI
Search by non-existent tag	searchByTag()	"quantum"	Returns an empty list
Delete existing record by URI	deleteByURI()	URI of an existing record	Record is deleted and removed from dataset
Delete non-existent URI	deleteByURI()	Random/invalid URI	"Record not found"
Sort dataset by creation date	sortByCreationDateDesc()	Dataset with records created at different times	Records returned in descending date order
Search by multiple tags (any match)	searchByTags()	["AI", "ethics"]	Returns records with at least one of the tags
Search by multiple tags (all must match)	searchByAllTags()	["php", "algorithms"]	Only records that contain all tags are returned

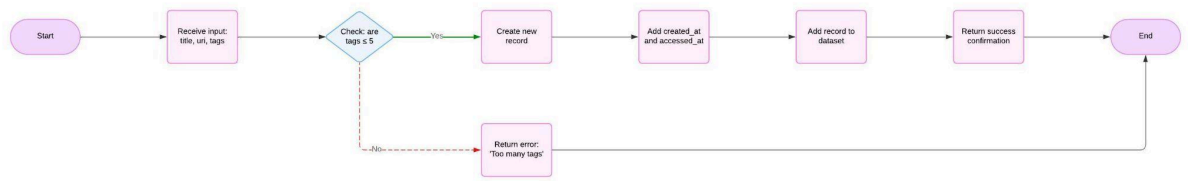
Edit existing record title and tags	editRecord()	URI of record, new title, new tag array	Record is updated accordingly
Edit record with invalid URI	editRecord()	Non-existent URI	"Record not found"
Search by keyword in title	searchByKeyword()	“Essex”	Returns records with “Essex” in title or URI
Search by uncommon word	searchByKeyword()	“gibberish”	Returns an empty list

Algorithms and Implementation

Below are the algorithms for the Interest Book, implemented in PHP without specialized functions, as required. Each is accompanied by a visual flowchart (see attached diagrams) and a brief description.

Add record

Adds a new record to `$dataset` if the tag count is ≤ 5. Uses a simple array append operation (Cormen et al., 2009).

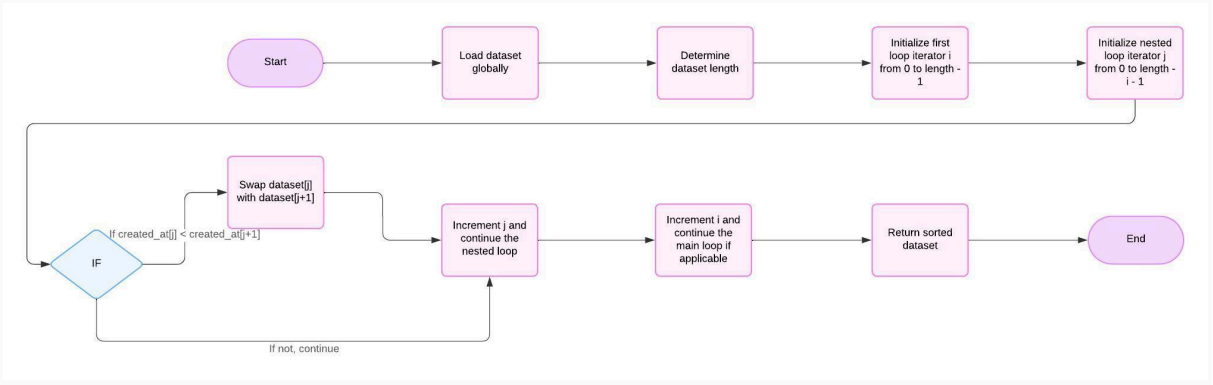


Textual Flowchart:

1. Start the function.
2. Receive input parameters: title, uri, tags[].
3. Check whether the number of tags exceeds five.
   - If yes: return the error message "Too many tags", then exit.
   - If no: proceed to the next step.
4. Create a new record containing:
   - title
   - uri
   - tags[]
   - created_at = current timestamp
   - accessed_at = current timestamp
5. Add the new record to the dataset.
6. Return a success confirmation message.
7. End the function.

Sort by creationDate

Uses bubble sort for descending order by created_at (Knuth, 1997).

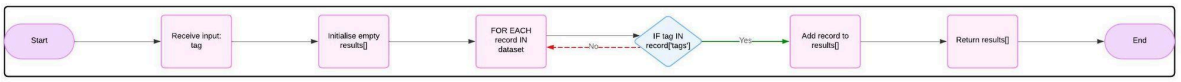


Textual Flowchart:

1. Start the function.
2. Access the global $dataset array containing all records.
3. Determine the number of records in $dataset and store it as length.
4. Begin an outer loop with variable i starting at 0, continuing while i is less than length - 1:
    - This loop controls the number of passes through the dataset.
5. For each value of i, begin an inner loop with variable j starting at 0, continuing while j is less than length - i - 1:
    - This loop compares adjacent records in the dataset.
6. For each value of j, compare the "created_at" timestamp of the record at index j with the "created_at" timestamp of the record at index j + 1:
    - If the timestamp at j is less than the timestamp at j + 1 (i.e., the record at j is older):
        - Store the record at index j in a temporary variable temp.
        - Replace the record at index j with the record at index j + 1.
        - Replace the record at index j + 1 with the contents of temp.
        - This swaps the two records to place the newer one first.
    - If the timestamp at j is not less than the timestamp at j + 1, do nothing and proceed to the next j.
7. After the inner loop completes, increment i and repeat step 4 until all passes are done.
8. Once the outer loop completes, the dataset is sorted in descending order by "created_at".
9. Return the sorted $dataset.

Search by tag (single)

Linearly searches tags in each record, returning matches



Textual Flowchart:

1. Start the function.
2. Receive input parameter: tag.
3. Initialise an empty results array.
4. Loop through each record in the dataset.
5. For each record, check if the provided tag exists in record["tags"].
   - If yes: add the record to the results array.
6. After all records have been checked, return the results array.
7. End the function.

Search by tags (match all)

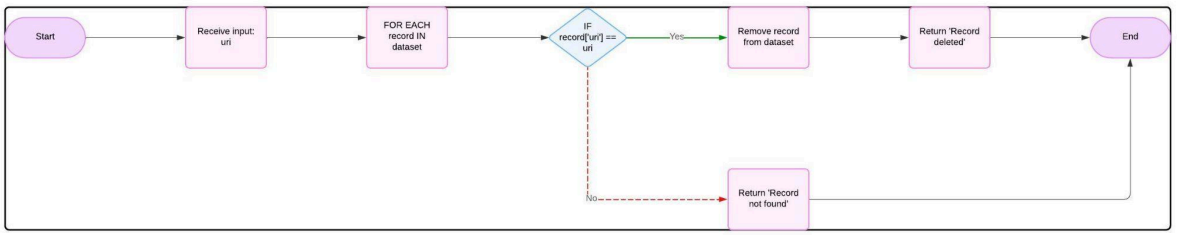
Ensures all tags match in a record.

Textual Flowchart:

1. Start the function.
2. Receive input parameter: tags[].
3. Initialise an empty results array.
4. Loop through each record in the dataset.
5. For each record:
   - Loop through each tag in tags[].
   - Check if the all tags exists in record["tags"].
     - If true: add the record to the results array and stop validating this record.
6. After all records have been processed, return the results array.
7. End the function.

Delete by URI

Removes a record by matching URI using linear search.

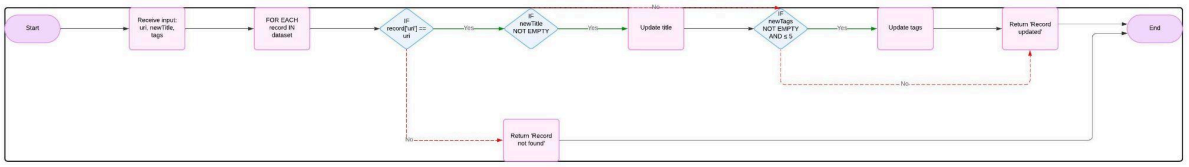


Textual Flowchart:

1. Start the function.
2. Receive input parameter: uri.
3. Loop through each record in the dataset.
4. For each record, compare record["uri"] with the input uri.
   - If they match: remove the record from the dataset and return "Record deleted", then end the function.
5. If no matching URI is found after checking all records, return "Record not found".
6. End the function.

Edit record

Updates a record by URI if conditions are met.

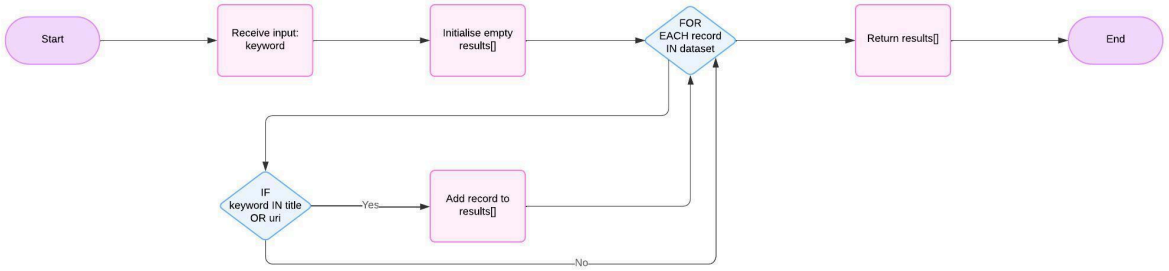


Textual Flowchart:

1. Start the function.
2. Receive input parameters: uri, newTitle, newTags[].
3. Loop through each record in the dataset.
4. For each record, check whether record["uri"] matches the input uri.
   - If no match, continue to the next record.
   - If a match is found:
     - If newTitle is not empty: update record["title"].
     - If newTags[] is not empty and contains five or fewer tags: update record["tags"].
     - Return "Record updated" and exit.
5. If no matching record was found, return "Record not found".
6. End the function.

Search by keyword

Searches title and tags for keywords (corrected from URI).



Textual Flowchart:

1. Start the function.
2. Receive input parameter: keyword.
3. Initialise an empty results array.
4. Loop through each record in the dataset.
5. For each record, perform the following checks:
   - Check if the keyword exists (case-insensitive match) in record["title"].
   - OR check if the keyword exists in record["tags"].
   - If either condition is true:
     - Add the record to the results array.
6. After all records have been evaluated, return the results array.
7. End the function.