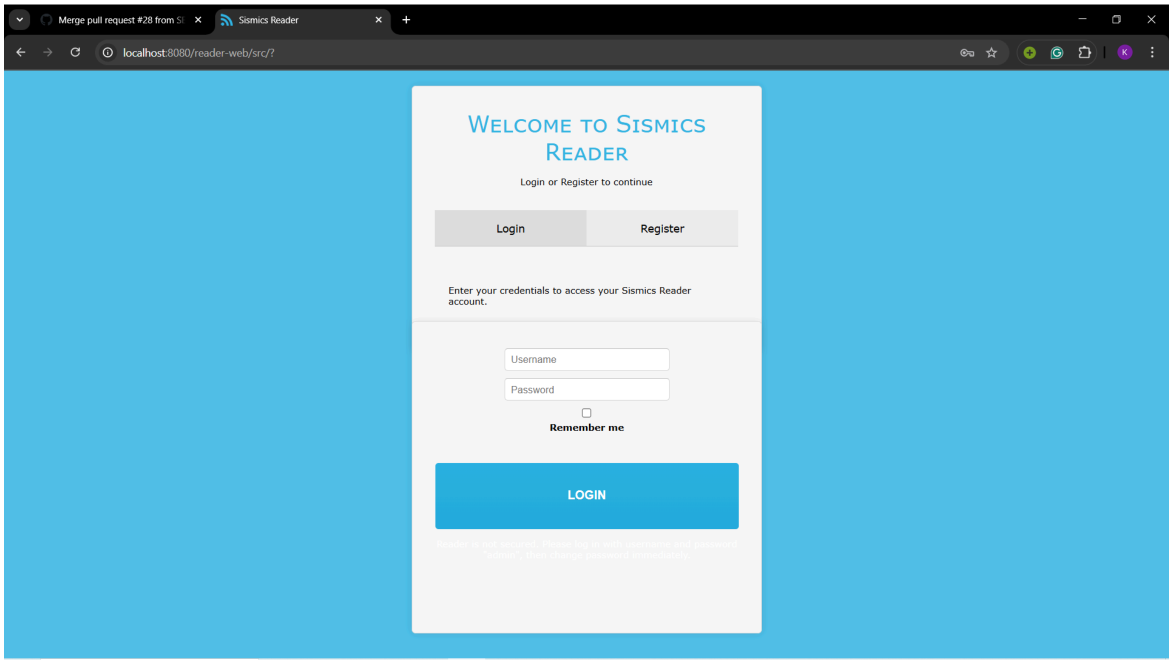
Project 2 Final Report

Task 1: User Registration



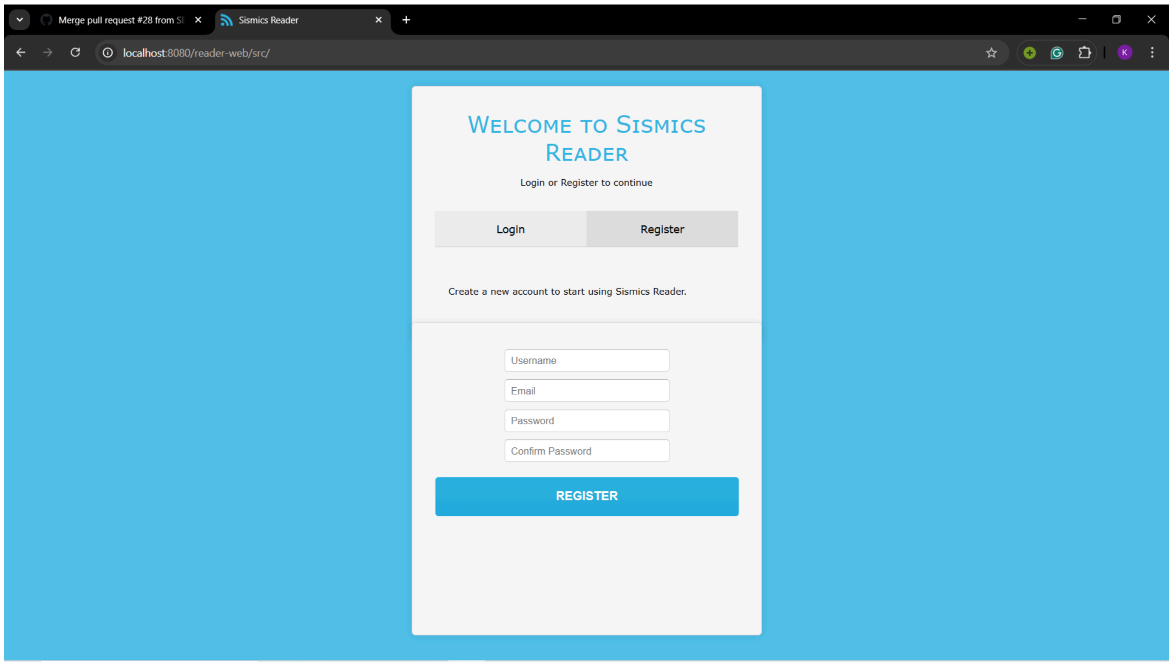
Merge pull request #28 from S
Sismics Reader
localhost:8080/reader-web/src/?
Login
Register
Username
Password
Remember me
LOGIN

Welcome to Sismics Reader

Login or Register to continue

Enter your credentials to access your Sismics Reader account.

Reader is not secured. Please log in with username and password "admin", then change password immediately.



Welcome to Sismics Reader

Login or Register to continue

Create a new account to start using Sismics Reader.

Task 1: User Registration

Task Description

The goal of this task is to introduce a new registration method that allows users to create their own accounts without requiring admin intervention. This enhances user experience by providing a frictionless sign-up process for the RSS Reader Service while also reducing the administrative burden.

Implementation Details

To achieve this, the following features were implemented:

1. A new registration form was added to the login page, allowing new users to sign up.
2. Users can now choose between logging in or registering for a new account.
3. Upon submitting the registration form, users can create an account without requiring admin approval.
4. Once registered, users can log in as standard users.
5. Admin account creation remains separate, and distinct registration functions have been implemented for users and admins.
6. The Factory Method design pattern was used to manage different user roles efficiently and ensure scalability.

Design Patterns Used

The **Factory Method** pattern was chosen to handle user creation due to its ability to manage different types of users with varying attributes. The benefits of using this pattern include:

- **Modularity**: It simplifies the addition of new user roles without modifying existing code.
- **Maintainability**: Encapsulating responsibilities enhances code organization and readability.
- **Scalability**: The system can accommodate future enhancements with minimal changes.

Changes in Files and Descriptions

The implementation required modifications and additions across multiple files:

Modified Files:

- **index.html**: Added a registration form to the login page.
- **r.user.js**: Implemented registration functionality and API calls for user registration.
- **r.util.js**: Added a new API path for user registration.
- **r.wizard.js**: Enabled form switching between login and registration.

- **main.less**: Updated CSS styling to accommodate the registration form.
- **UserResource.java**: Added a new registration method for frontpage registration and abstracted user creation to the factories.

Newly Created Files:

- **Application.java**: Implements the interface for utilizing the Factory Method.
- **UserRegistrationFactory.java**: Defines the abstract Factory Method for user creation.
- **SimpleUserRegistrationFactory.java**: Provides a concrete implementation of the Factory Method for standard users.
- **UserRegistration.java**: Implements the abstract product of user creation.
- **SimpleUserRegistration.java**: Provides a concrete implementation of user creation for normal users.

Conclusion

This implementation introduces a seamless user registration process, allowing new users to create accounts independently. Using the Factory Method ensures flexibility and maintainability, making it easier to extend user roles in the future.

Task 2: Filtering by RSS Feed

Task Description

The task is about having articles displayed from just a few subscriptions and/or categories instead of all the articles in the "All" section. This will allow the user to view only the content which he/she wants to view at that time, and also not juggle between multiple related categories or subscriptions.

Implementation Details

UI Changes

Here is how the UI looks like, with check boxes beside categories and subscriptions, which helps the user to select the categories and subscriptions to be included in their feeds.

Code Changes

This function added to r.subscription.js implements the behaviour of the check boxes.

This code snippet implements the filtering of articles to display in the feed.

Implementation description

Here are the points for implementation of this feature:

1. A checkbox was added beside each subscription and category the user has subscribed to or created.
2. The action of checkbox is to include or exclude certain subscriptions or categories from the view.
3. When ticked, the subscription or category content is shown in the all articles section.
4. When no checkbox is ticked, all articles from all subscriptions are shown by default.
5. Logic for selective subscription articles is as follows:
    a. The initial values for each subscription title is set to be false.
    b. If a subscription checkbox is ticked, its value is changed to true.
    c. All articles are fetched from the backend, and check is put while displaying it.
    d. The check is about the subscriptions of each article, if it has the value of true only then display.

Design Patterns Used

No design patterns were used for this specific task.

Changes in Files and Description

The changes were done in 2 files:

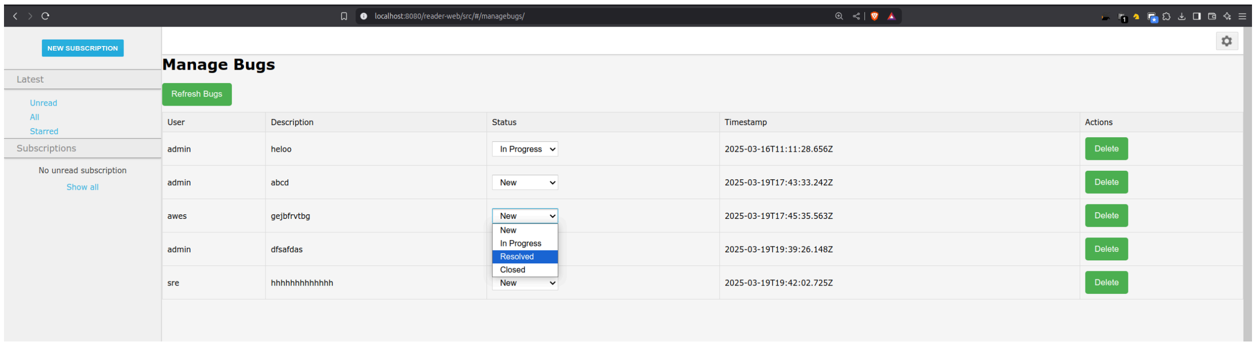
- **r.subscription.js:** UI was changed here, to add the checkbox for subscriptions and categories.
- r**.feed.js:** The filter was implemented here, to check for the subscription of each article before displaying in the feed.

Conclusion

This implementation helps the users to filter their feeds based on their interests, enabling more personalized browsing and content management. The user can set the feed interests based on selecting the categories or subscriptions using the checkbox.

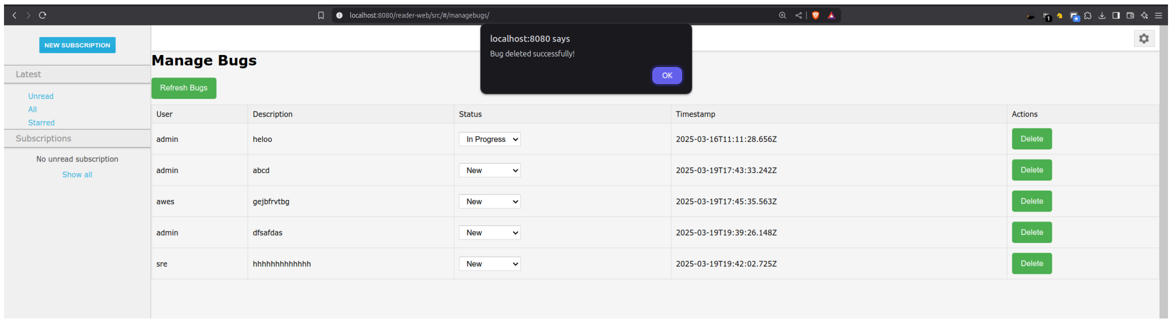
Task 3: Bug Reporting

1. Task Description



Manage Bugs

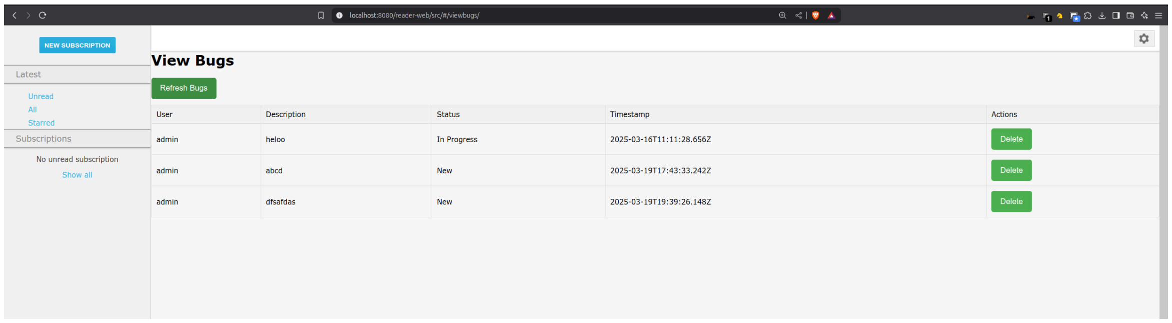
User	Description	Status	Timestamp	Actions
admin	heloo	In Progress	2025-03-16T11:11:28.656Z	Delete
admin	abcd	New	2025-03-19T17:43:33.242Z	Delete
awes	gejbfrvtbg	New	2025-03-19T17:45:35.563Z	Delete
admin	dfsafdas		2025-03-19T19:39:26.148Z	Delete
sre	hhhhhhhhhhhhh	New	2025-03-19T19:42:02.725Z	Delete



localhost:8080 says
Bug deleted successfully!

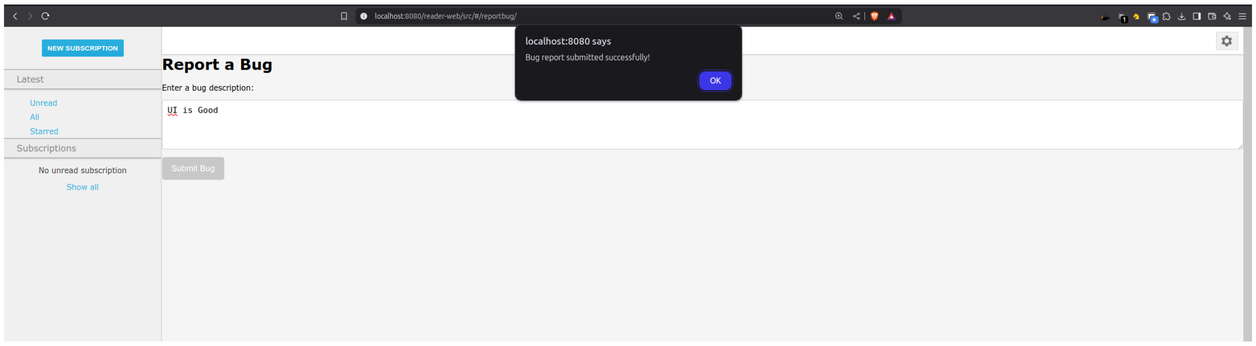
Manage Bugs

User	Description	Status	Timestamp	Actions
admin	heloo	In Progress	2025-03-16T11:11:28.656Z	Delete
admin	abcd	New	2025-03-19T17:43:33.242Z	Delete
awes	gejbfrvtbg	New	2025-03-19T17:45:35.563Z	Delete
admin	dfsafdas	New	2025-03-19T19:39:26.148Z	Delete
sre	hhhhhhhhhhhhh	New	2025-03-19T19:42:02.725Z	Delete



View Bugs

User	Description	Status	Timestamp	Actions
admin	heloo	In Progress	2025-03-16T11:11:28.656Z	Delete
admin	abcd	New	2025-03-19T17:43:33.242Z	Delete
admin	dfsafdas	New	2025-03-19T19:39:26.148Z	Delete



localhost:8080 says
Bug report submitted successfully!

Report a Bug

Enter a bug description:

UI is Good

3. Bug Reporting (15 Marks)

Bug reporting also needs an overhaul. At present, reporting an issue whisks users away to GitHub. While that's great for developers, it's not the best user experience.

Users will be able to submit issues within the RSS Reader itself - to make things simpler an issue is just a statement (description) with a timestamp. The admin will have access to a bug dashboard where they can view, mark bugs as resolved, or delete irrelevant reports. Users should also be able to delete their reported bugs (We imagine a few of you would like to do that for the doubts you send us).

2. Implementation Details

2.1 Overview

The RSS Reader was extended with an in-app bug reporting feature. This feature was integrated into the existing UI and backed by a new client-side JavaScript module (r.BugReport.js) and a corresponding server-side Java REST resource (ReportBugResources.java) using JAX-RS.

2.2 UI

New sections were added for reporting (reportbug-container), viewing (viewbugs-container), and managing bugs (managebugs-container). These sections are accessible via the settings menu and are navigated using URL routing (e.g., #/reportbug/). The UI includes:

- **Report a Bug:** A textarea (bugDescription) for entering the bug description and a submit button (bug-submit-button).
- **View Bugs:** A table (bugTable) displaying the user's submitted bugs, with a delete button for each bug.
- **Manage Bugs:** A table (manageBugTable) displaying all submitted bugs (for administrators), with a status dropdown (New, In Progress, Resolved, Closed) and a delete button for each bug.
- "Refresh bugs" buttons are included on the "View Bugs" and "Manage Bugs" views.

2.3 Client-Side Logic (r.BugReport.js)

The r.BugReport.js module handles user interactions, form submission, and AJAX requests to the server. Key functions include:

- **init:** Initializes event handlers for button clicks and table setup.
- **submitBug:** Collects the bug description, creates a bug report object (with User, Description, Status, and Timestamp), and sends it to the server via a POST request to /api/reportBug.
- **loadBugs:** Loads the bug reports for the current user (GET request to /api/reportBug/view) and displays them in the bugTable. Includes a delete button handler (deleteBugFromView).
- **loadManageBugs:** Loads all bug reports (GET request to /api/reportBug/manage) and displays them in the manageBugTable. Includes status update (updateBugStatus) and delete button (deleteBug) handlers.
- **updateBugStatus:** Updates the status of a bug (called when the status dropdown is changed in the Manage Bugs view). Sends a POST request to /api/reportBug/update with the updated bug information.
- **deleteBug:** Deletes a bug (called from the Manage Bugs view). Sends a POST request to api/reportBug/delete with the updated bug list.

- **deleteBugFromView:** Deletes a bug (called from the user's View Bugs). Gets *all* bugs, accounts for client-side filtering, then sends a delete request.

2.4 Server-Side Logic (ReportBugResources.java)

The ReportBugResources.java class provides the REST endpoints for managing bug reports:

- Uses a file-based storage mechanism (bug-db.json) to persist bug data. The file is located in the web application's root directory.
- Handles authentication using SecurityFilter and IPrincipal.
- **submitBug (POST):** Receives a new bug report, adds the username (if authenticated), and appends it to the bug-db.json file.
- **getBugs (GET):** Retrieves all bug reports (primarily for internal use; the frontend uses the /view and /manage endpoints).
- **getViewBugs (GET):** Retrieves only the bug reports submitted by the currently logged-in user.
- **updateBug (POST):** Updates the status of a bug, identified by its index. The *entire* updated bug list is sent from the client.
- **deleteBug (POST):** Deletes a bug. The updated list (after deletion on the client-side) is sent to the server.
- **getManageBugs (GET):** Retrieves all bug reports, but only if the logged-in user is "admin".

3. API Endpoints

Method	Endpoint	Description	Request Body	Response Body
POST	/api/reportBug	Submit a new bug report.	{ "User": "string", "Description": "string", "Status": "string", "Timestamp": "string" }	{ "message": "string" } or { "error": "string" }
GET	/api/reportBug	Get all bug reports (Used internally by server).	None	{ "bugs": [{ "User": "string", "Description": "string", "Status": "string", "Timestamp": "string", "index": "int" }, ...] }
GET	/api/reportBug/view	Get bug reports for the current user.	None	{ "bugs": [{ "User": "string", "Description": "string", "Status": "string", "Timestamp": "string" }, ...] }

GET	/api/reportBug/manage	Get all bug reports (admin only).	None	{ "bugs": [{ "User": "string", "Description": "string", "Status": "string", "Timestamp": "string", "index": "int" }, ...] }
POST	/api/reportBug/update	Update the status of a bug (admin). Expects the entire modified array from client.	{ "index": int, "status": "string", "bugs": [{ "User": "string", "Description": "string", "Status": "string", "Timestamp": "string"}, ...] }	{ "message": "string" } or { "error": "string" }
POST	/api/reportBug/delete	Delete a bug (user or admin). Expects the entire modified array from client.	{ "bugs": [{ "User": "string", "Description": "string", "Status": "string", "Timestamp": "string"}, ...] }	{ "message": "string" } or { "error": "string" }

4. Design Patterns Used

- This isn't exactly the **Observer pattern**, but it works similarly. When we click refresh or submit, the JavaScript function handles it first. If it's a **POST request**, it collects data and sends it to the backend. If it's **GET**, it just fetches the data. The backend processes the request, runs the right method, and sends a response. Then, our JavaScript updates the UI, usually with an alert confirming the action.

5. Changes in Files

File	Changes
index.html	Added UI elements: containers (reportbug-container, viewbugs-container, managebugs-container), textarea (bugDescription), buttons (bug-submit-button, refresh-bugs-button, refresh-manage-bugs-button), tables (bugTable, manageBugTable). Integrated with jQuery History.js for binding the URL.
r.BugReport.js	(New) Implements client-side logic for bug reporting: submitting, loading, updating, and deleting bugs. Uses AJAX.
r.main.js	Added lines to hide the new bug reporting containers on application load. and
ReportBugResources.java	(New) Implements the server-side REST endpoints for bug reporting (submission, retrieval, updates, deletion). Uses file-based storage (bug-db.json).

Task 4: Nested Categories in Feed Reader

Feature Overview

Making Categories Better (10 Marks)

The goal is to enhance the organization of categories by supporting nested structures. Users should be able to group feeds in a structured way, such as a "Technology" category containing subcategories like "AI," "Blockchain," or "Cybersecurity."

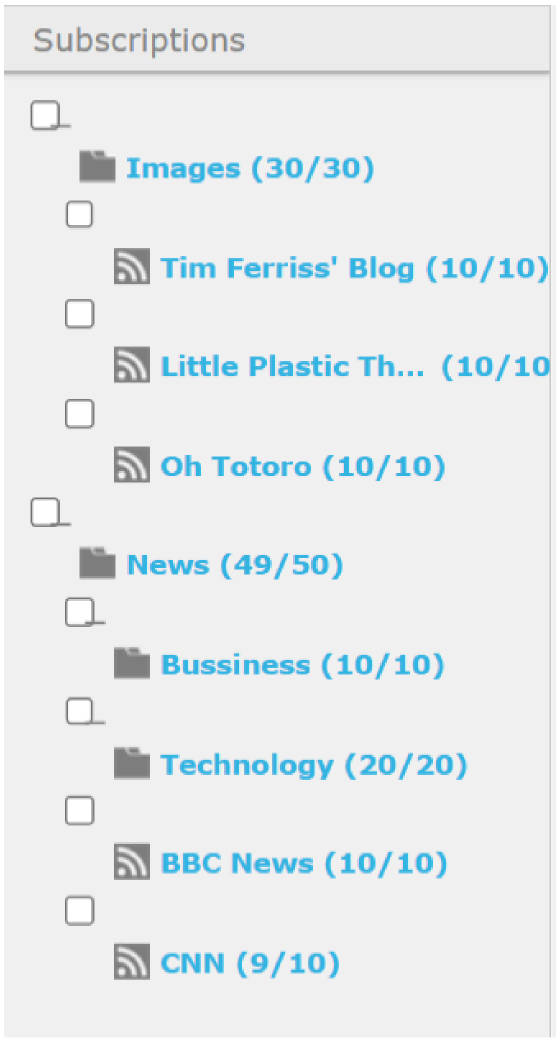
Key Requirements:

- Support up to **5 levels** of nested categories.
- Each category (and subcategory) should display metadata:
  - **Number of unread items**
  - **Total count of articles**
- UI should support **collapsible** nested categories.
- A category should host both **individual feeds** and **subcategories**.

Observations

- The **database structure already supports** nested categories using a **parent category ID** in the child category.
- The **UI currently does not support nested categories** (only flat categories are displayed).
- The function **findSubCategory** (CategoryDao) retrieves subcategories.
- The function **findByCriteria** (FeedSubscriptionDao) retrieves feed subscriptions.

UI changes

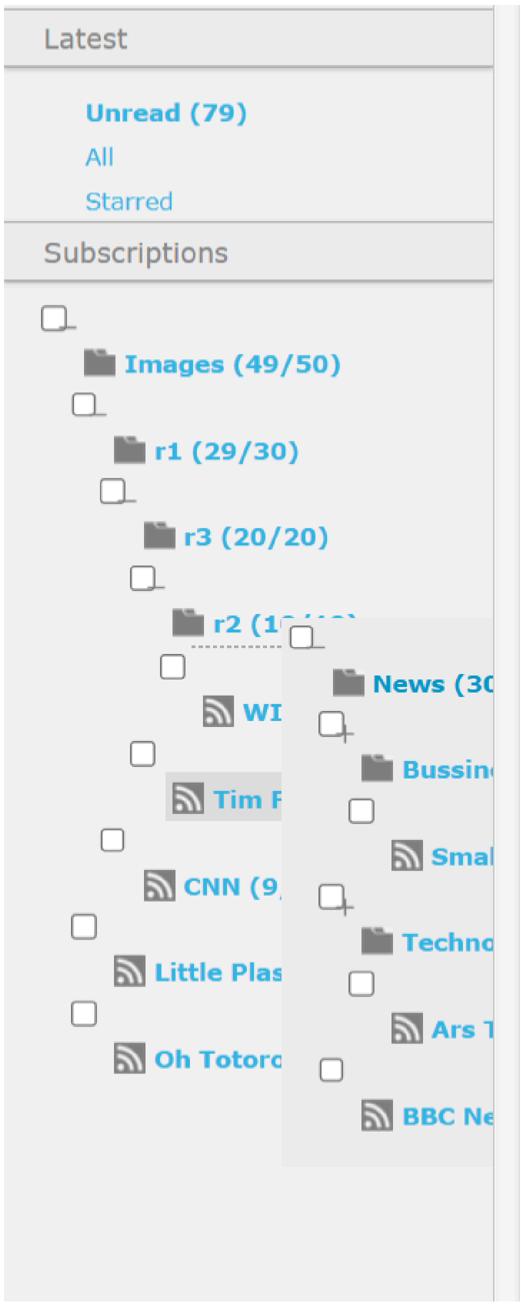


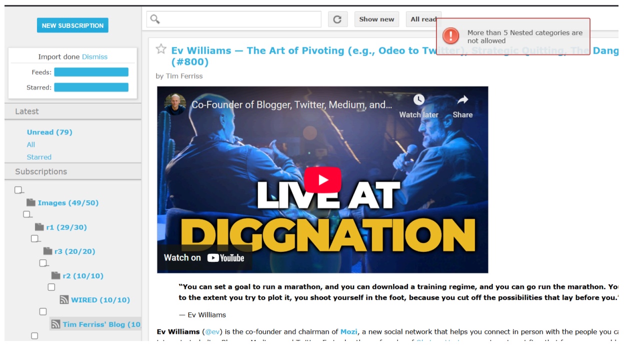
Subscriptions

- Images (30/30)
  - Tim Ferriss' Blog (10/10)
  - Little Plastic Th... (10/10
  - Oh Totoro (10/10)
- News (49/50)
  - Bussiness (10/10)
  - Technology (20/20)
  - BBC News (10/10)
  - CNN (9/10)

Here, u can see News contains BBC and CNN subs as well as business and technology categories as well

U can see the unread and total articles for categories as well as subs.





More than 5 Nested categories are not allowed

Dragging and dropping news items with two levels into the R2 category is blocked, as categories can only be nested up to five levels deep.

Changes Implemented (Step-by-Step)

1. Backend Updates

Database & DAO Layer Changes

- **Updated** `CategoryDao.java` :
    - Added support for **retrieving nested categories recursively**.
    - Introduced **validation for parent-child relationships** to prevent circular references and enforce the 5-level depth limit.
- **Updated** `FeedSubscriptionDao.java` :
    - Created `updateTotalCount()` to track **total article count** per subscription.

API & Resource Layer Changes

- **Modified** `SubscriptionResource.java` :
    - Updated `list()` function to **return a nested JSON structure** instead of a flat list.
    - Implemented `buildCategoryJsonRecursive()` to recursively construct the nested structure of categories and feeds.
- **Updated** `CategoryResource.java` :
    - Enhanced the `update()` method to **validate nesting depth** and **handle dynamic parent changes**.
    - Prevented circular references.
- **Added New API in** `AllResource.java` :
    - `/count` **API** retrieves the total number of articles for a given feed.

2. Frontend Updates

Building Nested UI Elements

- **Modified** `r.subscription.js` :
    - **Updated** `update()` to handle **nested category structures** dynamically.
    - Used a **recursive function** to generate collapsible UI elements representing categories and feeds.
    - Integrated metadata (unread count and total count of articles) into the UI.
    - Implemented **drag-and-drop** functionality for reordering categories and feeds while enforcing a **5-level nesting limit**.
    - Restricted invalid parent assignments and prevented circular category relationships.

Enhancing User Interaction

- **Updated** `initSorting()` :
    - Enabled **nested drag-and-drop sorting** for categories and feeds.

    - Dynamically updates **parent ID** when items are moved.
    - Prevents invalid moves beyond the allowed depth, displaying an error message.
- **Updated `message.en.js & message.ko.js`:**
    - Added **localized error messages** for exceeding depth limits and invalid nesting.

Summary

This implementation ensures that categories are **hierarchical**, **collapsible**, and **metadata-rich** while preventing excessive nesting and invalid configurations. The changes are reflected across the **backend, UI, and database layers**, ensuring a seamless user experience.

Changes in Files and Description

File Path	Description of Changes and Overall Functionality
reader-web/src/main/webapp/src/javascripts/r.subscription.js	Refactored the subscription update logic to handle nested categories and subscriptions, added drag-and-drop functionality, and improved error handling and logging.
reader-web/src/main/java/com/sismics/reader/rest/resource/SubscriptionResource.java	Refactored the `list` method to build JSON responses recursively for categories and subscriptions, added error handling, and improved drag-and-drop functionality for nested categories and subscriptions.
reader-core/src/main/java/com/sismics/reader/core/dao/jpa/FeedSubscriptionDao.java	Introduced `updateTotalCount` method to update the number of total articles in a user subscription.
reader-core/src/main/java/com/sismics/reader/core/dao/jpa/dto/FeedSubscriptionDto.java	Added `totalUserArticleCount` field with getter and setter methods to store the total number of articles in a user subscription.
reader-core/src/main/java/com/sismics/reader/core/dao/jpa/CategoryDao.java	Added logging and error handling to the `create`

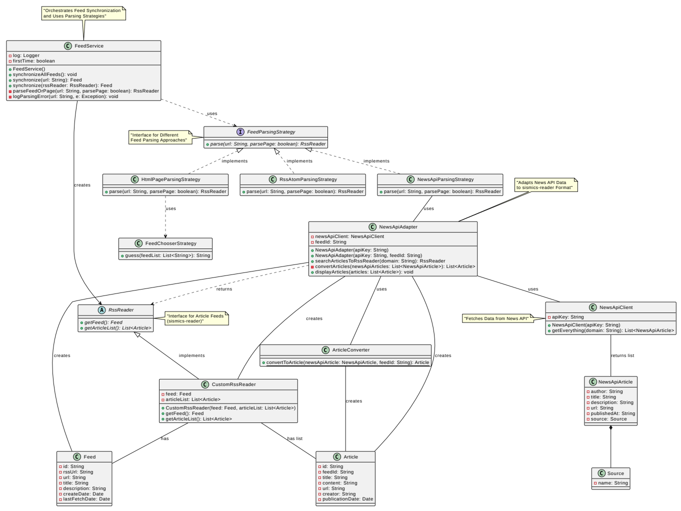
	method, and updated validation rules for required fields in category creation. Enhanced the `update` method to handle parent category updates.
reader-web/src/main/java/com/sismics/reader/rest/resource/AllResource.java	Added an API endpoint to get the count of articles in a feed by `feedId`.
reader-web/src/main/java/com/sismics/reader/rest/resource/CategoryResource.java	Enhanced the `update` method to handle parent category updates and added validation for nested category depth to prevent categories from being nested more than 5 levels deep.
reader-web/src/main/webapp/src/locales/messages.en.js	Updated messages to include error handling for nested categories exceeding the maximum depth of 5 levels.

Task 5a: News API Integration

Task Description

Some web pages lack RSS feeds. Our current system can only extract content from pages that have RSS feeds. This task adds a new feature to the system, enabling it to extract content using the NewsAPI, format it as an RSS feed, and present it to the user. This is achieved through a single API call per feed.

Implementation Details


"Orchestrates Feed Synchronization and Uses Parsing Strategies"
FeedService
log: Logger
firstTime: boolean
FeedService()
synchronizeAllFeeds(): void
synchronize(url: String): Feed
synchronize(rssReader: RssReader): Feed
parseFeedOrPage(url: String, parsePage: boolean): RssReader
logParsingError(url: String, e: Exception): void
uses
creates
"Interface for Different Feed Parsing Approaches"
FeedParsingStrategy
parse(url: String, parsePage: boolean): RssReader
implements
implements
implements
HtmlPageParsingStrategy
parse(url: String, parsePage: boolean): RssReader
RssAtomParsingStrategy
parse(url: String, parsePage: boolean): RssReader
NewsApiParsingStrategy
parse(url: String, parsePage: boolean): RssReader
"Adapts News API Data to sismics-reader Format"
uses
uses
FeedChooserStrategy
guess(feedList: List<String>): String
NewsApiAdapter
newsApiClient: NewsApiClient
feedId: String
NewsApiAdapter(apiKey: String)
NewsApiAdapter(apiKey: String, feedId: String)
searchArticlesToRssReader(domain: String): RssReader
convertArticles(newsApiArticles: List<NewsApiArticle>): List<Article>
displayArticles(articles: List<Article>): void
returns
uses
uses
creates
creates
creates
RssReader
getFeed(): Feed
getArticleList(): List<Article>
"Interface for Article Feeds (sismics-reader)"
implements
NewsApiClient
apiKey: String
NewsApiClient(apiKey: String)
getEverything(domain: String): List<NewsApiArticle>
"Fetches Data from News API"
ArticleConverter
convertToArticle(newsApiArticle: NewsApiArticle, feedId: String): Article
CustomRssReader
feed: Feed
articleList: List<Article>
CustomRssReader(feed: Feed, articleList: List<Article>)
getFeed(): Feed
getArticleList(): List<Article>
returns list
creates
NewsApiArticle
author: String
title: String
description: String
url: String
publishedAt: String
source: Source
has
has list
Feed
id: String
rssUrl: String
url: String
title: String
description: String
createDate: Date
lastFetchDate: Date
Article
id: String
feedId: String
title: String
content: String
url: String
creator: String
publicationDate: Date
Source
name: String

Users can now type the link of the website from where they would like to get articles for the feed. Even if this website does not have the RSS Feed format, we now use NewsApi to support other websites which can be scraped using NewsApi.

To achieve this, a new directory called "NewsApp" was created, containing all the functionality to get the news from the website.

- The **Adapter pattern** was used to convert the format returned by NewsApi to our RssReader format.
- The **Strategy pattern** was implemented to decide whether to use the existing RssReader class or the NewsApi-based approach when given a URL.
- The fetched feed and articles are stored in the database, allowing any user to subscribe to these feeds readily, instead of making a call to the website every time someone wants to subscribe to this feed. This reduces the number of Api calls required.

2.1 Overview

The RSS Reader was extended with a feature to fetch and parse articles from News API and convert them to a custom RSS feed format. This feature allows the application to dynamically create RSS feeds from articles fetched via News API, providing a seamless integration of external news content. The implementation leverages the Adapter design pattern to integrate the News API client and the Commander pattern for handling the conversion and display logic.

2.4 Server-Side Logic

The following classes and files were updated or added to implement this feature:

- **pom.xml**: Dependencies for HTTP client, JSON parsing, and logging were added.
- **CustomRssReader.java**: A new class for representing custom RSS readers.
- **NewsApiAdapter.java**: A new class that acts as an adapter to fetch and convert News API articles to RSS format.

- **NewsApiClient.java**: A new class that handles HTTP requests to the News API.
- **NewsApiArticle.java**: A new class representing the structure of News API articles.
- **ArticleConverter.java**: A new utility class for converting News API articles to the application's `Article` format.
- **RssReader.java**: The existing RSS reader class was modified to accommodate the new feed types and elements.
- **FeedService.java**: The existing service class was updated to integrate the News API adapter.
- **FeedParsingStrategy.java**: Defines the strategy to be used to select between the RssReader and the newsApiClient to process the URL

Design Patterns Used

Adapter Pattern

News API Integration using the Adapter Pattern

Introduction

This report details the implementation of a `NewsApiAdapter` class to facilitate the integration of the News API as a news source within an application utilizing the `sismics-reader` library. The adapter pattern was chosen to bridge the gap between the News API's data structure and the format expected by the `sismics-reader` system.

1. Advantages of the Adapter Pattern

The adapter pattern offers several key advantages in software development:

- **Decoupling:** It isolates the application's core logic from the specific implementation details of external APIs or libraries. Changes to the external source require modifications only within the adapter, minimizing impact on the rest of the system.
- **Reusability:** The adapter can be reused across multiple components or applications that need to access the same external data source, ensuring consistency and reducing code duplication.
- **Maintainability:** By centralizing the integration logic within the adapter, maintenance and updates become significantly easier. Modifications to handle API changes or new features are localized, preventing widespread code changes.
- **Testability:** Adapters can be easily mocked during unit testing, allowing developers to test the application's logic independently of the external API's availability or behavior. This ensures robust and reliable testing.

2. Rationale for the Adapter Pattern in this Scenario

The adapter pattern is particularly well-suited for this scenario due to the following factors:

- **Incompatible Data Formats:** The News API returns news articles in a custom `NewsApiArticle` format. The `sismics-reader` library, however, expects articles to conform to its own `com.sismics.reader.core.model.jpa.Article` data structure. An adapter is essential to perform the necessary data transformation.

- **Need for a Specific Interface:** The `sismics-reader` library utilizes an `RssReader` interface to consume article feeds. The adapter is responsible for presenting the data retrieved from the News API in a format compliant with the `RssReader` contract, which uses the `Feed` and `Article` objects defined in `sismics-reader`.
- **Abstraction of API Complexity:** The adapter encapsulates the complexities of interacting with the News API, including authentication, request formatting, and error handling. This simplifies the application's code and improves its overall clarity.
- **Flexibility for Future Changes:** Using the adapter allows for easy switching to different news sources in the future. A new adapter could be written for a different API while preserving the core application logic.

3. Input and Output Formats

To clarify the data transformation performed by the `NewsApiAdapter`, the following table outlines the mapping of features from the input `NewsApiArticle` format (as received from the News API) to the output `Article` format (as required by the `sismics-reader` library):

Feature Category	NewsApiArticle (Input - JSON from News API)	Article (Output - sismics-reader JPA Entity)	Notes
Identification			
Unique Identifier	(Implicit - based on URL & other fields)	`id` (String, UUID)	The adapter (or `ArticleConverter`) generates a unique ID for each `Article`. Could be a hash of the URL and title.
Feed Association	(None)	`feedId` (String)	The `feedId` is explicitly set in the adapter to associate the article with the correct feed (obtained from the News API source).
Content			
Title	`title` (String)	`title` (String)	The title of the news article.
Content/Description	`description` (String)	`content` (String)	The adapter maps the News API's `description` field to the `Article`'s `content` field. Further processing (e.g., fetching the full article content) may be needed to fully populate the content.
URL	`url` (String)	`url` (String)	The URL of the original news article.
Metadata			
Author/Creator	`author` (String)	`creator` (String)	The author of the article. May be null or "unknown" if not provided by the News API.
Publication Date	`publishedAt` (String - ISO 8601)	`publicationDate` (Date)	The adapter converts the ISO 8601 date string from the News API into a Java `Date` object.

Source Name	`source.name` (String)	(Potentially derived and stored elsewhere)	The source name from the News API (e.g., "Reuters", "BBC News"). This might be used for categorization or filtering, but isn't directly mapped to an `Article` field.
Additional Information			
Image URL	`urlToImage` (String)	(Not Directly Mapped)	URL to the article's image; this field is not directly mapped, but it could be saved into a custom field if needed.
Other Fields	(Various - e.g., `content`)	(Potentially custom fields)	Additional fields from the News API response can be stored in custom fields within the `Article` entity if required.

Strategy Pattern

Advantages of using Strategy Pattern

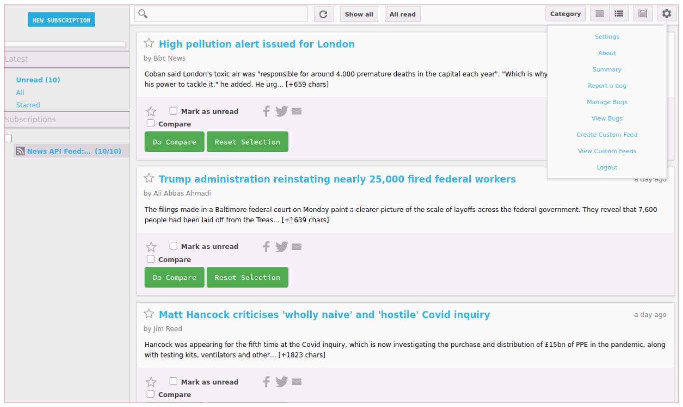
- **Flexibility:** The decision to use `NewsApiAdapter` or RssReader is at runtime. This enables the application to dynamically adapt to the specific characteristics of each URL. If certain RSS feeds work better than the News Api, the application will automatically prefer using those.
- **Extensibility:** New ways to handle website links (for example, using a web crawler) can be integrated without modifying the core code.
- **Maintainability:** Strategy Pattern promotes loose coupling.

Changes in Files and Description

File Path	Functionality
`reader-core/src/main/java/com/newsreader/CustomRssReader.java`	New file created. Custom implementation of `RssReader` to handle feeds and articles.
`reader-core/src/main/java/com/newsreader/NewsApiAdapter.java`	New file created. Implements an adapter `NewsApiAdapter` to fetch and convert articles from News API into `RssReader`.
`reader-core/src/main/java/com/newsreader/NewsApiClient.java`	New file created, Implements `NewsApiClient` to interact with the News API and fetch articles.
`reader-core/src/main/java/com/newsreader/model/NewsApiArticle.java`	New file created. Defines the `NewsApiArticle` model class to map News API article data.
`reader-core/src/main/java/com/newsreader/util/ArticleConverter.java`	New file created. Provides utility methods to convert News API articles into the application's `Article` format.
`reader-core/src/main/java/com/sismics/reader/core/dao/file/rss/RssReader.java`	added integration to parse RSS/Atom feeds and handle various feed elements.
`reader-core/src/main/java/com/sismics/reader/core/service/FeedService.java`	added integration to use `NewsApiAdapter` or `RssReader` strategically for fetching articles

Task 5b: Custom Feed Creation

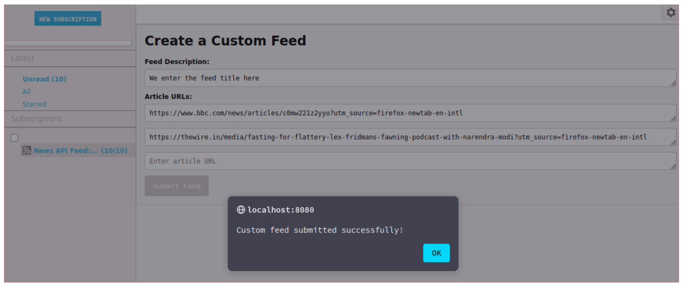
Task Description



High pollution alert issued for London
by Bbc News
Coban said London's toxic air was "responsible for around 4,000 premature deaths in the capital each year". "Which is why his power to tackle it," he added. He urg... [+659 chars]

Trump administration reinstating nearly 25,000 fired federal workers
by Ali Abbas Ahmadi
The filings made in a Baltimore federal court on Monday paint a clearer picture of the scale of layoffs across the federal government. They reveal that 7,600 people had been laid off from the Treas... [+1639 chars]

Matt Hancock criticises 'wholly naive' and 'hostile' Covid inquiry — a day ago
by Jim Reed
Hancock was appearing for the fifth time at the Covid inquiry, which is now investigating the purchase and distribution of £15bn of PPE in the pandemic, along with testing kits, ventilators and other... [+1823 chars]

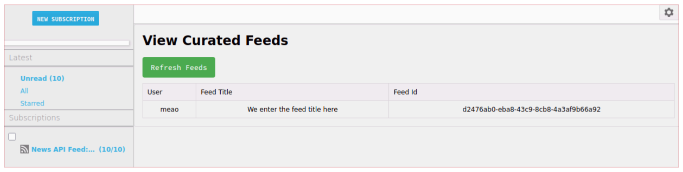


Create a Custom Feed

Feed Description:
We enter the feed title here

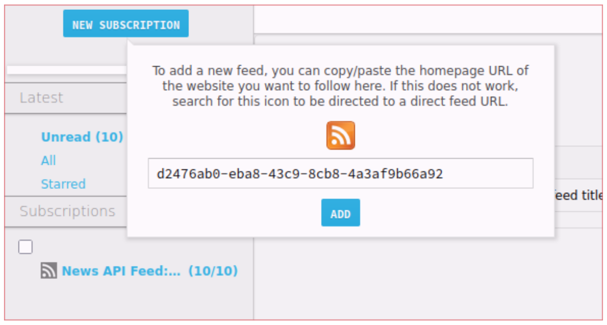
Article URLs:
https://www.bbc.com/news/articles/c0mw221z2yyo?utm_source=firefox-newtab-en-intl
https://thewire.in/media/fasting-for-flattery-lex-fridmans-fawning-podcast-with-narendra-modi?utm_source=firefox-newtab-en-intl
Enter article URL

localhost:8080
Custom feed submitted successfully!



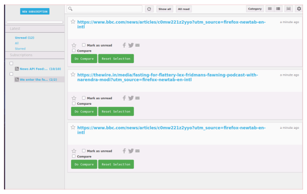
View Curated Feeds

User	Feed Title	Feed Id
meao	We enter the feed title here	d2476ab0-eba8-43c9-8cb8-4a3af9b66a92



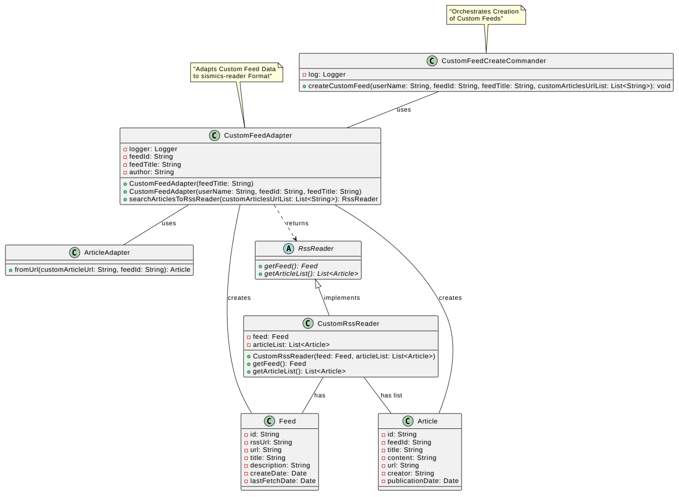
To add a new feed, you can copy/paste the homepage URL of the website you want to follow here. If this does not work, search for this icon to be directed to a direct feed URL.

d2476ab0-eba8-43c9-8cb8-4a3af9b66a92



Users should be able to curate their own feeds by selecting articles from multiple sources. Other users should be able to subscribe to these curated feeds.

Implementation Details


"Orchestrates Creation of Custom Feeds"
CustomFeedCreateCommander
log: Logger
createCustomFeed(userName: String, feedId: String, feedTitle: String, customArticlesUrlList: List<String>): void
uses
"Adapts Custom Feed Data to sismics-reader Format"
CustomFeedAdapter
logger: Logger
feedId: String
feedTitle: String
author: String
CustomFeedAdapter(feedTitle: String)
CustomFeedAdapter(userName: String, feedId: String, feedTitle: String)
searchArticlesToRssReader(customArticlesUrlList: List<String>): RssReader
uses
returns
ArticleAdapter
fromUrl(customArticleUrl: String, feedId: String): Article
RssReader
getFeed(): Feed
getArticleList(): List<Article>
creates
implements
creates
CustomRssReader
feed: Feed
articleList: List<Article>
CustomRssReader(feed: Feed, articleList: List<Article>)
getFeed(): Feed
getArticleList(): List<Article>
has
has list
Feed
id: String
rssUrl: String
url: String
title: String
description: String
createDate: Date
lastFetchDate: Date
Article
id: String
feedId: String
title: String
content: String
url: String
creator: String
publicationDate: Date

2.1 Overview

The RSS Reader was extended with a feature to allow users to curate their own feeds with articles they want to add. Other users can also view these custom feeds and subscribe to them. The implementation uses the Adapter and Commander design patterns.

2.2 UI

New sections were added for creating (customfeed-container) and viewing custom feeds (viewcustomfeeds-container). These sections are accessible via the settings menu and are navigated using URL routing (e.g., #/customfeed/). The UI includes:

- **Create a Custom Feed:** Textareas for entering the feed description (feed_title) and article URLs (article_url1, article_url2), and a submit button (feed-submit-button).
- **View Custom Feeds:** A table (customfeedstable) displaying all curated feeds, with a "Refresh Feeds" button (refresh-custom-feeds-button).

2.3 Client-Side Logic (r.customfeed.js)

The r.customfeed.js module handles user interactions, form submission, and AJAX requests to the server. Key functions include:

- **init:** Initializes event handlers for button clicks and table setup.
- **submitFeed:** Collects the feed description and article URLs, creates a feed object, and sends it to the server via a POST request to /api/customFeed.
- **loadCustomFeeds:** Loads all custom feeds (GET request to /api/customFeed/view) and displays them in the customfeedstable. Includes a delete button handler (deleteFeed).

2.4 Server-Side Logic (CustomFeedResources.java)

The CustomFeedResources.java class provides the REST endpoints for managing custom feeds:

- Uses a file-based storage mechanism (custom-feed-db.json) to persist feed data. The file is located in the web application's root directory.
- Handles authentication using SecurityFilter and IPrincipal.
- **submitFeed (POST):** Receives a new custom feed, adds the username (if authenticated), and appends it to the custom-feed-db.json file.
- **getCustomFeeds (GET):** Retrieves all custom feeds.
- **getViewCustomFeeds (GET):** Retrieves custom feeds curated by the currently logged-in user.
- **deleteFeed (POST):** Deletes a custom feed. The updated list (after deletion on the client-side) is sent to the server.

3. API Endpoints

Method	Endpoint	Description	Request Body	Response Body
POST	/api/customFeed	Submit a new custom feed.	{ "User": "string", "Description": "string", "ArticleURLs": ["string"], "Timestamp": "string" }	{ "message": "string" } or { "error": "string" }
GET	/api/customFeed	Get all custom feeds.	None	{ "feeds": [{ "User": "string", "Description": "string", "ArticleURLs": ["string"], "Timestamp": "string", "index": "int" }, ...] }
GET	/api/customFeed/view	Get custom feeds for the current user.	None	{ "feeds": [{ "User": "string", "Description": "string", "ArticleURLs": ["string"], "Timestamp": "string" }, ...] }
POST	/api/customFeed/delete	Delete a custom feed. Expects the	{ "feeds": [{ "User": "string", "Description":	{ "message": "string" } or { "error": "string" }

		entire modified array from client.	"string", "ArticleURLs": ["string"], "Timestamp": "string"}, ...] }	

Design Patterns Used

- **Command Pattern:** The `CustomFeedCreateCommander` class acts as the invoker, receiving a request to create a custom feed and then delegating the actual creation process to other components.
- **Adapter Pattern:** The `CustomFeedAdapter` class adapts the custom article URLs and feed metadata into the `sismics-reader` system's expected `RssReader` format.

Rationale for Design Pattern Choices

- **Command Pattern:**
    - **Purpose:** The Command pattern encapsulates a request as an object, thereby allowing for parameterizing clients with different requests, queueing or logging requests, and supporting undoable operations.
    - **Why It's Appropriate:** In the context of custom feed creation, the process involves several steps: validating user input, fetching article data, creating a new feed, and saving the feed to the database. The `CustomFeedCreateCommander` encapsulates this entire process as a command.
        - **Decoupling:** It decouples the UI (which initiates the feed creation request) from the actual logic of creating and persisting the feed. The UI doesn't need to know the details of how the feed is created; it just needs to tell the commander to execute the command.
        - **Centralized Control:** It provides a central point to manage the feed creation process, allowing for easy addition of new steps (e.g., sending a notification to the user after the feed is created) or modifications to existing steps.
        - **Extensibility:** If you wanted to add features like undoing a feed creation or scheduling feed creations, the Command pattern would make those features easier to implement.
    - **Analogy:** Think of the `CustomFeedCreateCommander` as a remote control. The user presses a button ("create custom feed"), and the remote control (commander) sends a signal to the appropriate device (the creation logic) to perform the action. The user doesn't need to know how the device works, just how to use the remote.
- **Adapter Pattern:**
    - **Purpose:** The Adapter pattern allows classes with incompatible interfaces to work together. It acts as a translator between two different interfaces.
    - **Why It's Appropriate:** The custom feed creation process involves taking data from a user (feed title and article URLs) and transforming it into a format that the `sismics-reader` system can understand and persist. Specifically, it needs to create an `RssReader` object with the appropriate `Feed` and `Article` objects.

    - **Interface Compatibility:** The user-provided data doesn't directly match the structure of the `Feed` and `Article` objects. The `CustomFeedAdapter` takes this raw data and adapts it to fit the required interface. It takes the list of article URLs and converts them into a list of `Article` objects, and it constructs the `Feed` object with the appropriate metadata.
    - **Abstraction:** The adapter hides the complexity of the data transformation from the rest of the system. The `FeedService` (which actually persists the feed) doesn't need to know where the data came from or how it was transformed; it just receives an `RssReader` object in the expected format.
  - **Benefit:** This approach isolates change. If the data coming from the user changes (e.g., if a new field is added), only the adapter needs to be updated. The `FeedService` and other parts of the system remain unaffected.

Input and Output Formats for CustomFeedAdapter

The following table outlines the transformation performed by the `CustomFeedAdapter`, detailing how user-provided data and the `ArticleAdapter` output are converted into the required `Feed` and `Article` objects for the `sismics-reader` library.

Feature Category	User Input / ArticleAdapter Output	Article (Output - sismics-reader JPA Entity)	Feed (Output - sismics-reader JPA Entity)	Notes
Feed Identification				
Unique Feed Identifier	`feedId` (String, User Provided or Generated)	`feedId` (String)	`id` (String)	The `feedId` comes from the `CustomFeedAdapter` constructor, can be user-provided or autogenerated. It is used to tie articles to the feed.
Feed Content				
Feed Title	`feedTitle` (String, User Provided)		`title` (String)	The `feedTitle` comes directly from the user input.
Feed Description/Author	`author` (String, User Provided or "Anonymous")		`description` (String)	The `author` comes from the `CustomFeedAdapter` constructor, either user-provided or "Anonymous".
Feed RSS URL	`feedId` (String)		`rssUrl` (String)	The RSS URL is set to the feed ID so to be compatible with the system
Feed URL	`feedId` (String)		`url` (String)	The URL is set to the feed ID so to be compatible with the system
Article Details				

Article URL (Input to ArticleAdapter)	`customArticlesUrlList` (List<String>, User Provided)			The Article urls provided by the user goes into the ArticleAdapter.
Article ID (ArticleAdapter Output)	(Implicit - URL and Feed ID used)	`id` (String)		The `ArticleAdapter` generates an ID for the article
Article Feed ID (ArticleAdapter Output)	`feedId` (String)	`feedId` (String)		The `ArticleAdapter` links the article to the feed.
Article URL (ArticleAdapter Output)	Article URL	`url` (String)		The `ArticleAdapter` saves the article URL.
Article Title (ArticleAdapter Output)	Article Title	`title` (String)		The `ArticleAdapter` derives the title from the article

Changes in Files and Description

File Path	Description of Changes and Overall Functionality
`reader-core/src/main/java/com/newsreader/NewsApiAdapter.java`	**Modified**: Added functionality to handle empty article list by returning an empty `RssReader`. The overall functionality is to adapt the News API responses to be compatible with the application's RSS reader format.
`reader-core/src/main/java/com/sismics/reader/core/constant/Constants.java`	**Modified**: Added a new constant `CONFIG_NEWS_API_KEY` for the News API key configuration. The file contains various constants used across the application.
`reader-core/src/main/java/com/sismics/reader/core/customfeeds/ArticleAdapter.java`	**New**: This file was added to the repository. The overall functionality is to adapt individual articles for custom feeds.
`reader-core/src/main/java/com/sismics/reader/core/customfeeds/CustomFeedAdapter.java`	**New**: This file was added to the repository. The overall functionality is to adapt

	custom feeds to be compatible with the existing feed structures.
`reader-core/src/main/java/com/sismics/reader/core/customfeeds/CustomFeedCreateCommander.java`	**New**: This file was added to the repository. The overall functionality is to handle the creation commands for custom feeds.
`reader-core/src/main/java/com/sismics/reader/core/dao/jpa/FeedDao.java`	**Modified**: Added a method `getByFeedId` to retrieve an active feed by its ID. The file manages feed data access operations.
`reader-core/src/main/java/com/sismics/reader/core/model/context/AppContext.java`	**Modified**: Imported `CustomFeedCreateCommander`. The file provides global application context.
`reader-core/src/main/java/com/sismics/reader/core/service/FeedService.java`	**Modified**: Added a method to synchronize feeds and handle articles accordingly. The file manages feed-related services.
`reader-web/src/main/java/com/sismics/reader/rest/resource/CustomFeedResource.java`	**New**: This file was added to the repository. The overall functionality is to provide RESTful resources to manage custom feeds.
`reader-web/src/main/java/com/sismics/reader/rest/resource/SubscriptionResource.java`	**Modified**: Added new methods and print statements for debugging. The file handles subscription-related REST resources.
`reader-web/src/main/webapp/src/index.html`	**Modified**: Added sections for creating and viewing custom feeds, along with corresponding CSS styles. The file is the main HTML page for the web application.
`reader-web/src/main/webapp/src/javascripts/r.customfeed.js`	**New**: This file was added to the repository. The overall functionality is to manage custom feed-related JavaScript functions.

`reader-web/src/main/webapp/src/javascripts/r.main.js`	**Modified**: Included new containers for custom feeds in the reset function. The file manages the main JavaScript functions for the web application.
`reader-web/src/main/webapp/src/javascripts/r.subscription.js`	**Modified**: Added a console log for debugging purposes. The file manages subscription-related JavaScript functions.

Task 6a: Daily Summaries

Task Description

This feature helps get a summary of the unread articles of the user. The summary is a weighted summary, where articles in the higher levels are given more wieghtage than the ones in the lower levels.

Implementation Details

1. A new entry was added in the Settings tab where the user can view the summary
2. The unread articles from all the categories and subscriptions were fetched and based on their levels their summaries were asked to be generated from the array.
3. Gemini is prompted with the suitable prompt and the summary is retrieved. Following this, the result is displayed on the page

Design Patterns Used

No design patterns were used for this task, as it involved simple prompting

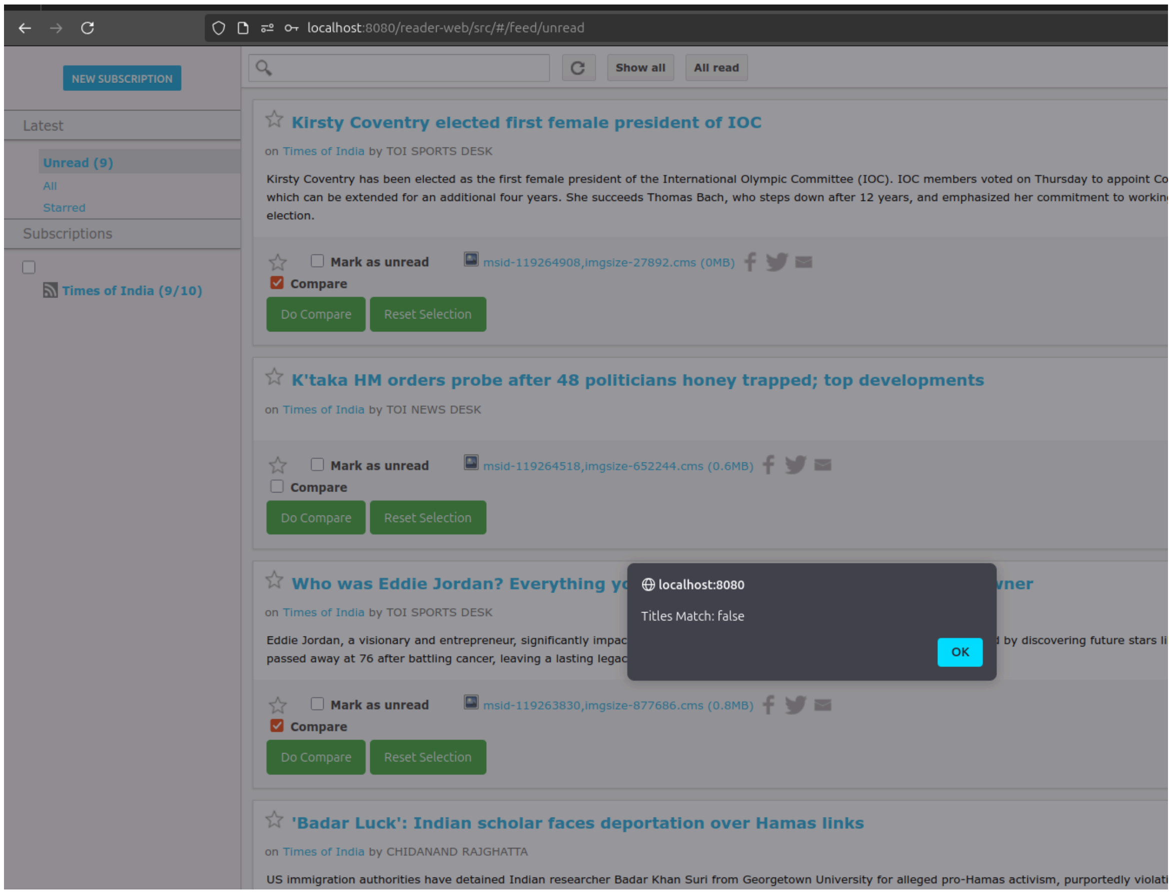
Changes in Files and Description

SummaryResource.java file has been added with all the logic to fetch the unread articles, process the query to be passed to the LLM and get the response

r.subscription.js file has been added to make a call to SummaryResource and display the summary in the UI.

Task 6b: Duplicate Detection

Duplicate Detection Feature


localhost:8080/reader-web/src/#/feed/unread

NEW SUBSCRIPTION

Show all · All read

Latest

Unread (9)

All

Starred

Subscriptions

Times of India (9/10)

Kirsty Coventry elected first female president of IOC

on Times of India by TOI SPORTS DESK

Kirsty Coventry has been elected as the first female president of the International Olympic Committee (IOC). IOC members voted on Thursday to appoint Co
which can be extended for an additional four years. She succeeds Thomas Bach, who steps down after 12 years, and emphasized her commitment to workin
election.

Mark as unread · msid-119264908,imgsize-27892.cms (0MB)

[✓ Compare]

Do Compare · Reset Selection

K'taka HM orders probe after 48 politicians honey trapped; top developments

on Times of India by TOI NEWS DESK

Mark as unread · msid-119264518,imgsize-652244.cms (0.6MB)

Compare

Do Compare · Reset Selection

Who was Eddie Jordan? Everything y … **wner**

on Times of India by TOI SPORTS DESK

Eddie Jordan, a visionary and entrepreneur, significantly impac … by discovering future stars li
passed away at 76 after battling cancer, leaving a lasting legac

localhost:8080

Titles Match: false

OK

Mark as unread · msid-119263830,imgsize-877686.cms (0.8MB)

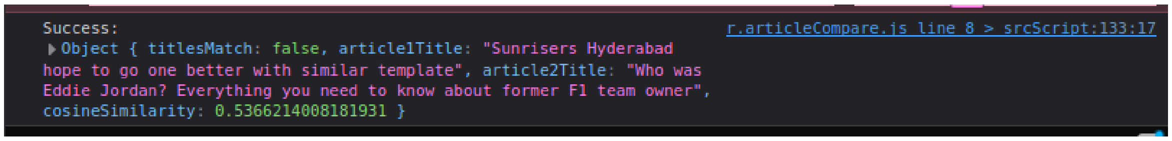
[✓ Compare]

Do Compare · Reset Selection

'Badar Luck': Indian scholar faces deportation over Hamas links

on Times of India by CHIDANAND RAJGHATTA

US immigration authorities have detained Indian researcher Badar Khan Suri from Georgetown University for alleged pro-Hamas activism, purportedly violati


```
Success:                                          r.articleCompare.js line 8 > srcScript:133:17
▶Object { titlesMatch: false, article1Title: "Sunrisers Hyderabad
hope to go one better with similar template", article2Title: "Who was
Eddie Jordan? Everything you need to know about former F1 team owner",
cosineSimilarity: 0.5366214008181931 }
```

1. Task Description

6B (5 Marks) Duplicate Detection

And while we're on the topic of automation, let's introduce duplicate detection. If two articles are essentially the same, we should be able to tell.

The system should be able to detect duplicate articles using simple techniques such as Named Entity Recognition (NER) and keyword-based similarity matching or semantic matching (your call). The implementation should avoid excessive computation and storage overhead. We expect the user to select the two articles for comparison in this task.

2. Implementation Details

2.1 Overview

The RSS Reader was extended with an article duplicate detection feature. This feature was integrated into the existing UI and backed by a new client-side JavaScript module (r.ArticleCompare.js) and a corresponding server-side Java REST resource (CompareArticleResource.java) using JAX-RS.

2.2 UI

Checkboxes (compareCheckbox) were added to each article's footer within the feed-item template. "Do Compare" (doCompareButton) and "Reset Selection" (resetSelectionButton) buttons were also added to the feed-item footer to trigger the comparison process and clear selections.

2.3 Client-Side Logic (r.ArticleCompare.js)

The r.ArticleCompare.js module handles the client-side logic:

- **init:** Sets up event listeners using event delegation for checkbox changes and button clicks.
- **updateComparedArticles:** Adds or removes articles from the comparedArticles array when checkboxes are toggled. Extracts article data (title, description, URL, etc.).
- **extractUrlFromElement:** Reliably extracts the article URL from different possible locations within the HTML.
- **handleCompareClick:** When the "Do Compare" button is clicked, sends the comparedArticles data to the server via a POST request to /api/compareArticle. Handles success and error responses.
- **resetComparison:** Resets comparison.

2.4 Server-Side Logic (CompareArticleResource.java)

The CompareArticleResource.java class provides the REST endpoint for article comparison:

- **compareArticles (POST):**
  - Receives a JSON array containing data for two articles.
  - Extracts the titles.
  - Uses the **Google Gemini API** to get text embeddings for the titles (semantic matching).
  - Calculates the **cosine similarity** between the embeddings.
  - Determines if the titles match based on a similarity threshold (0.8).
  - Returns a JSON response indicating the match result, titles, and similarity score.
  - Includes error handling for invalid JSON, missing titles, and Gemini API communication failures.
- **getEmbedding (private helper method):** Makes a request to the Google Gemini API to get a text embedding.
- **cosineSimilarity (private helper method):** Calculates the cosine similarity between two vectors.

3. API Endpoints

Method	Endpoint	Description	Request Body	Response Body
POST	/api/compareArticle	Compare two articles for duplication.	[{ "title": "string", "description": "string", ... }, { "title": "string", "description": "string", ... }]	{ "titlesMatch": boolean, "article1Title": "string", "article2Title": "string", "cosineSimilarity": number } or { "message": "string" }

4. Design Patterns Used

- **Observer Pattern:** This pattern is key to how we handle duplicate detection. In our setup, **r.ArticleCompare.js** acts as the *observer*, while the "Compare" checkboxes in each **feed-item** are the *subjects*. We use event delegation to listen for **change** events on all checkboxes, even those added dynamically as we scroll.

  At first, we tried storing selected articles in the modules that load them, but switching subscriptions wiped out the selections. To fix this, we created **r.ArticleCompare.js** to *observe* checkbox changes and keep a live list of selected articles (**comparedArticles**). This way, selections persist even when we switch subscriptions. The **updateComparedArticles** function acts as the *update* method in the Observer pattern, running every time a checkbox changes.

5. Changes in Files

File	Changes
index.html	Added "Compare" checkboxes (compareCheckbox), "Do Compare" (doCompareButton), and "Reset Selection" (resetSelectionButton) buttons to the feed-item template.
r.ArticleCompare.js	(New) Implements client-side logic for duplicate detection: managing article selections, extracting data, and sending comparison requests.
r.main.js	Added lines to hide containers on application load.
r.Article.js	Modified to ensure that the "Compare" checkbox states are preserved when switching between different subscriptions or views.
r.user.js	Modified to handle on click event while application boots
CompareArticleResource.java	(New) Implements the server-side REST endpoint for article comparison, using the Google Gemini API for semantic similarity.

Contribution Table

Task 1	Krishna Chaitanya

Task 2	Sankalp Bahad
Task 3	Yash Bhaskar
Task 4	Sreenivas
Task 5 (a & b)	Sreyas
Task 6a	Sreenivas & Sankalp
Task 6b	Yash Bhaskar