



PROJECT

ADVANCED TOPICS IN SOFTWARE DEVELOPMENT

GROUP 22

GROUP MEMBERS:

Angela Gilhotra	B00786846
Khushboo Patel	B00878765
Kavan Patel	B00869224
Rushi Patel	B00886157
Sarthak Patel	B00912612
Vivek Patel	B00874162

Faculty- Professor Tushar Sharma

Table of Contents

1. Overview of application.....	Error! Bookmark not defined.
2. Dependencies	Error! Bookmark not defined.
3. Build and deployment.....	4
4. Usage Scenario.....	5
5. Use Case Diagram.....	23
6. ER Diagram.....	24
7. Progress Chart	28
7. Contribution	26
8. References.....	31

Project Documentation: Smarticle

Overview of the application:

Smarticle is useful for anyone who wants to access the article via the web app and read the article of their choice. The application will allow users to post and like articles. As long as the user is not logged in, they can only see the public articles, the top five tweets according to the tags associated with each article, and the number of likes. Moreover, users can create their tags while posting an article, can filter the articles by authors, and categories, and can sort articles by date and number of likes. Upon registration, all features will be accessible to the user such as according to their tag preferences, the user can access both public and private articles. Furthermore, users can like or dislike an article. Apart from this, users can also update their profile, view their posted articles, and tweet count for each tag they added to their preference list.

Technologies used:

1. Backend: Spring Boot, JPA
2. Frontend: NextJS
3. Database: MySQL
4. Hosted on: Heroku

Dependencies:

Source code available on <https://git.cs.dal.ca/courses/2022-winter/csci-5308/group22>

Clone the project from the git (URL mentioned above).

The following are the Internal Dependencies used for building the project

- Spring boot JPA
- Spring boot Web
- MySQL Connector
- Spring boot Tomcat
- Jasypt Spring boot
- Spring Context
- Spring boot mail
- Spring boot thyme leaf
- Spring Security Crypto
- Jupyter Junit
- Jackson databind
- Spring boot Devtool
- Spring security core
- JWT

- Twitter4J
- HTTP Client
- OkHttp
- JSON
- Node.js
- JDK 11

Plugins

- Spring boot Maven Plugin
- Maven Compiler

Build and deployment:

1. Clone the project
2. Go to the frontend directory
3. In the terminal and execute the following command given below. This will host the application on localhost port 3000.

For Building the frontend

For installing the node module packages: `npm i`

```
D:\postarticle\group22\frontend>npm i
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated @fortawesome/fontawesome-common-types@0.3.0: Please upgrade to 6.1.0. https://fontawesome.com/docs/changelog/
npm WARN deprecated @fortawesome/fontawesome-common-types@0.3.0: Please upgrade to 6.1.0. https://fontawesome.com/docs/changelog/
npm WARN deprecated @material-ui/core@4.12.3: You can now upgrade to @mui/material. See the guide: https://mui.com/guides/migration-v4/

added 580 packages, and audited 581 packages in 35s

108 packages are looking for funding
  run `npm fund` for details

4 vulnerabilities (3 moderate, 1 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
npm notice
npm notice New patch version of npm available! 8.5.0 -> 8.5.5
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.5.5
npm notice Run npm install -g npm@8.5.5 to update!
npm notice
```

Figure 1: Installing node packages

For running the project on a local server: `npm run dev`

```

D:\postarticle\group22\frontend>npm run dev

> dev
> next dev

ready - started server on 0.0.0.0:3000, url: http://localhost:3000
event - compiled client and server successfully in 7.4s (359 modules)
Attention: Next.js now collects completely anonymous telemetry regarding usage.
This information is used to shape Next.js' roadmap and prioritize features.
You can learn more, including how to opt-out if you'd not like to participate in this anonymous program, by visiting the following URL:
https://nextjs.org/telemetry

wait - compiling / (client and server)...
event - compiled client and server successfully in 1297 ms (479 modules)

```

Figure 2: Frontend build successfully

For Building the backend

1. Go to the backend folder. Write the following command to build the project.
mvn clean install



```

[INFO] Building jar: D:\postarticle\group22\backend\target\smarticle-0.0.1-SNAPSHOT.jar
[INFO]
[INFO]      spring-boot-maven-plugin:2.6.3:repackage                @ smarticle
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO]      maven-install-plugin:2.5.2:install                      @ smarticle
[INFO] Installing D:\postarticle\group22\backend\target\smarticle-0.0.1-SNAPSHOT.jar to C:\Users\Bhavin\
.m2\repository\com\smarticle\smarticle\0.0.1-SNAPSHOT\smarticle-0.0.1-SNAPSHOT.jar
[INFO] Installing D:\postarticle\group22\backend\pom.xml to C:\Users\Bhavin\m2\repository\com\smarticl
e\smarticle\0.0.1-SNAPSHOT\smarticle-0.0.1-SNAPSHOT.pom
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 15.879 s
[INFO] Finished at: 2022-04-02T18:49:14-03:00
[INFO]
Bhavin@LAPTOP-CQRN9JVG MINGW64 /d/postarticle/group22/backend (main)
$

```

Figure 3: Backend application builds successfully

2. Build the project by setting the goal as **spring-boot:run**

Open the <http://localhost:3000> URL in the browser to run the project on the local machine.

Usage scenario

The following are the detailed uses for the application.

1. New users can register for the application by providing a valid and unique email id and a unique username that any user has not taken. After registering user has to verify the smarticle account by clicking on the verification link sent to the registered email id.

The screenshot shows a web browser window with the address bar displaying "smarticledigital.herokuapp.com/signup". The page title is "Signup | Smarticle". The main heading is "Signup". Below the heading, there are input fields for "User Name", "First Name", "Last Name", "Email", "Password", and "Confirm Password". The "User Name" field contains "Kavan1619", "First Name" contains "Kavan", "Last Name" contains "Patel", and "Email" contains "kavancanada19@gmail.com". The "Password" and "Confirm Password" fields are filled with dots. A green checkmark icon is visible next to the "Email" field. A notification box at the bottom of the form states "Verification link sent to registered email id".

Figure 4: Registering the new user

2. Validation is done on the signup page.

The screenshot shows the same "Signup" page, but with validation errors. The "User Name" field has a red error message "Provide a user name". The "First Name" field has a red error message "Enter your first name". The "Last Name" field has a red error message "Enter your last name". The "Email" field has a red error message "Enter your Email ID". The "Password" field has a red error message "Create a password for your account". The "Confirm Password" field has a red error message "Confirm password". Below the form, there are buttons for "Signup" and "Reset", and a link for "Already registered?".

Figure 5: Validation check for sign up page

The screenshot shows a web browser window with the address bar displaying "smarticledigital.herokuapp.com/signup". The page title is "Signup | Smarticle". The main heading is "Signup". Below it, there is a "User Name" label and a text input field containing "Kavan". A red error message below the field reads: "Use 8 characters or more for your username".

Figure 6: Validation check for username on the signup page

The screenshot shows the "Password" and "Confirm Password" fields on the Signup page. Both fields contain masked text (dots). A red error message below the fields reads: "Those passwords didn't match. Please try again."

Figure 7: Validation check for password and confirm password on the signup page

The screenshot shows the full Signup form with the following fields filled: "User Name" (KavanPatel2708), "First Name" (Kavan), "Last Name" (Patel), "Email" (kavancanada19@gmail.com), "Password" (masked), and "Confirm Password" (masked). A red error message below the "Email" field reads: "An account is already registered with the emailId".

Figure 8: Throws error if user adds email ID which is already registered in the database

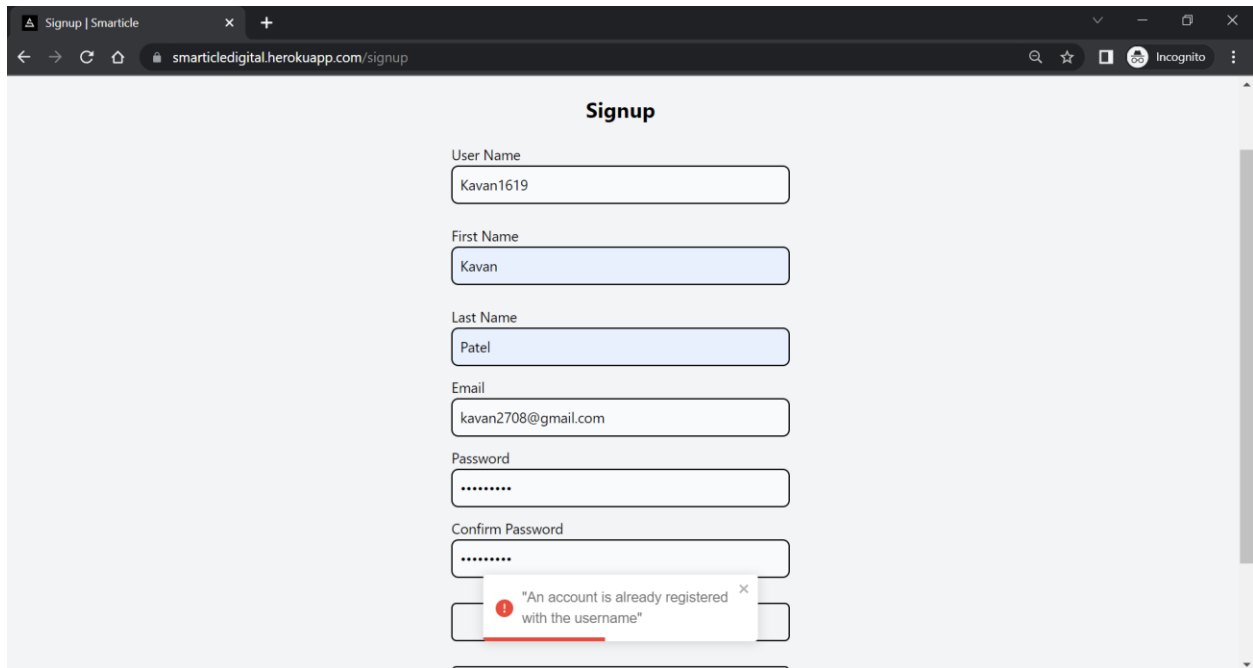


Figure 9: Throws error if the user adds the username which is already registered in the database

3. User needs to verify the account once registration is done successfully. The verification link will be sent to the email ID provided at the time of registration.

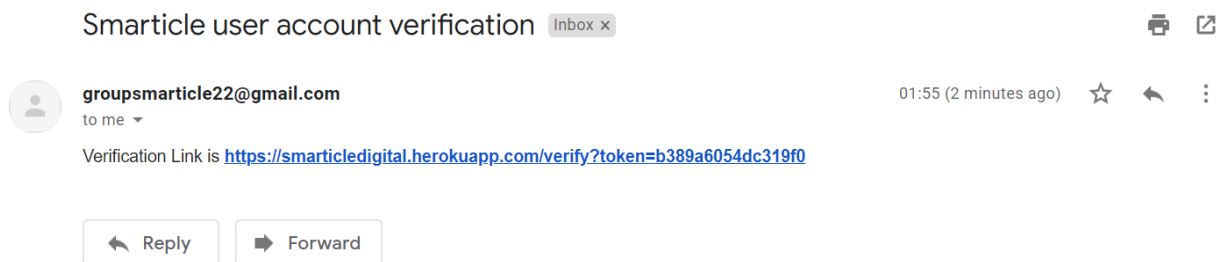


Figure 10: Verification link sent to the email

4. Once the user clicks the link the verification is done successfully.

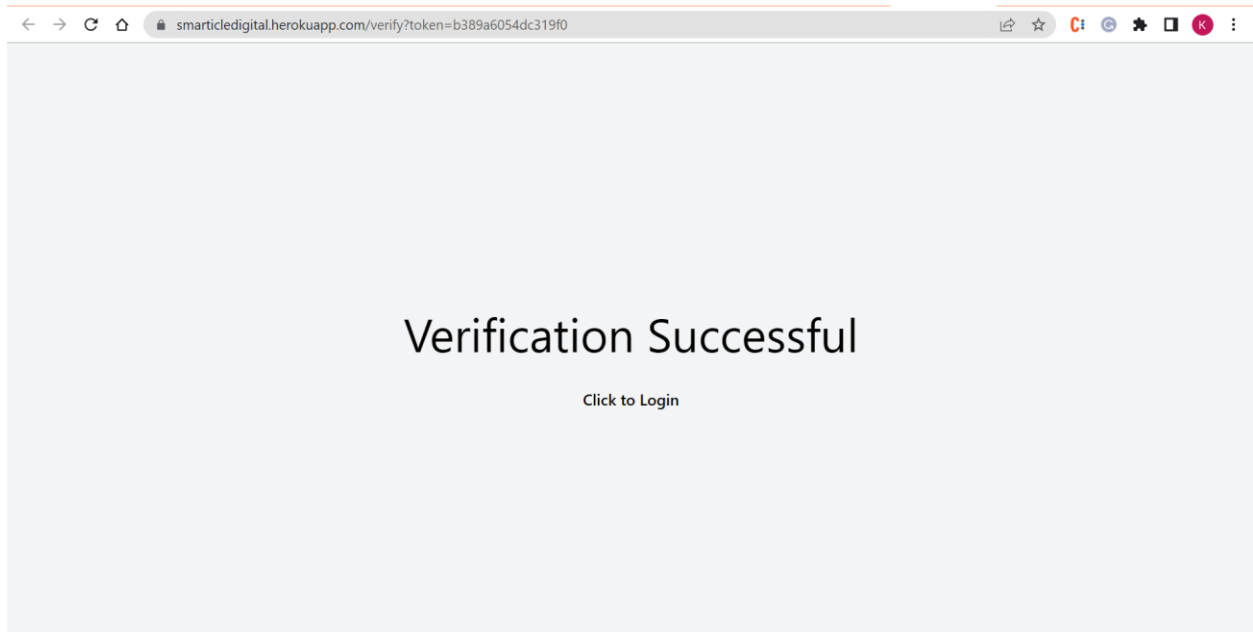


Figure 11: Verification is done successfully

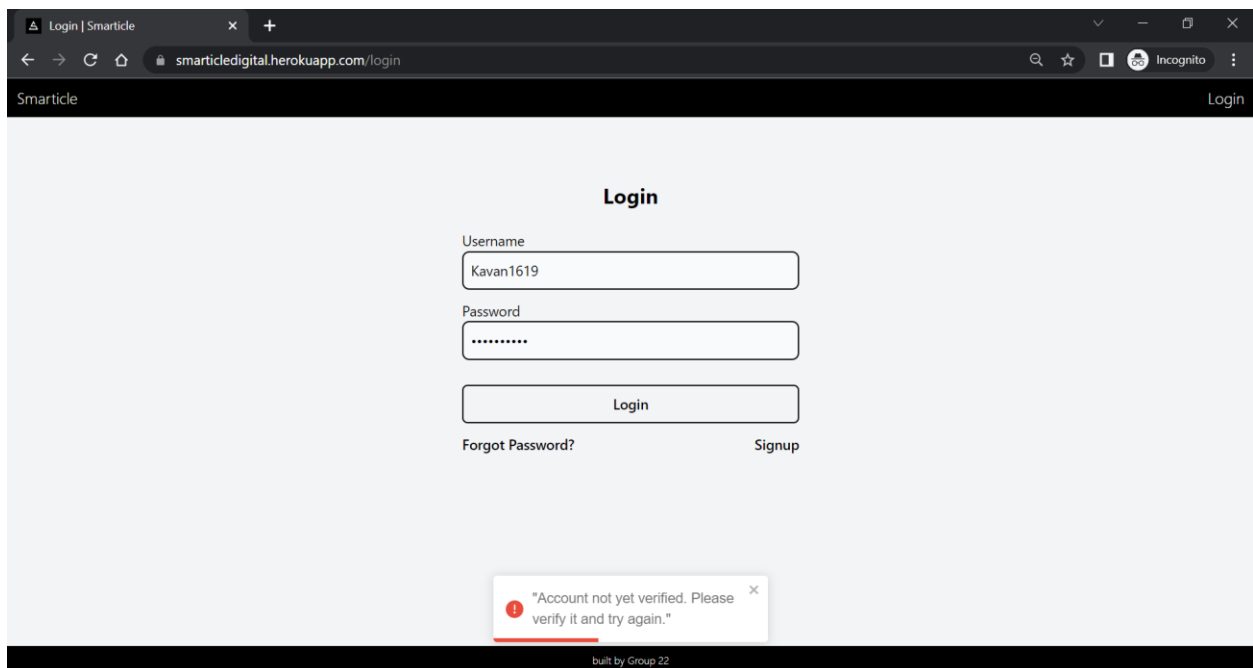


Figure 12: If verification is not done by the user

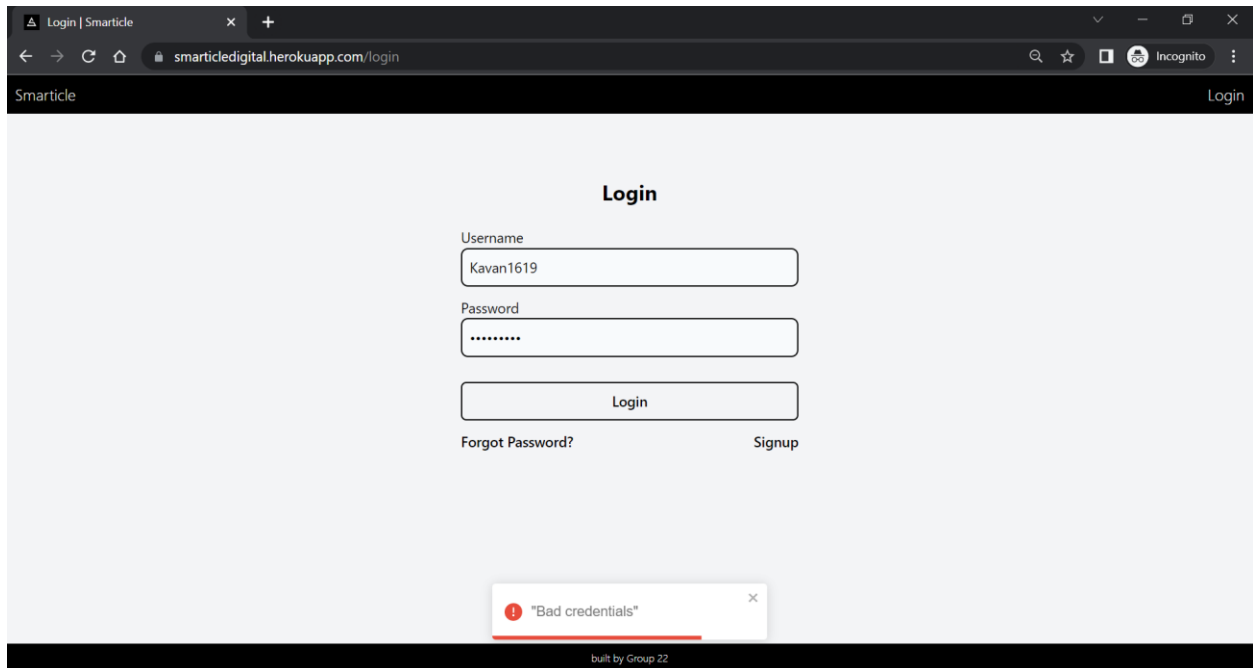


Figure 13: Invalid Credentials

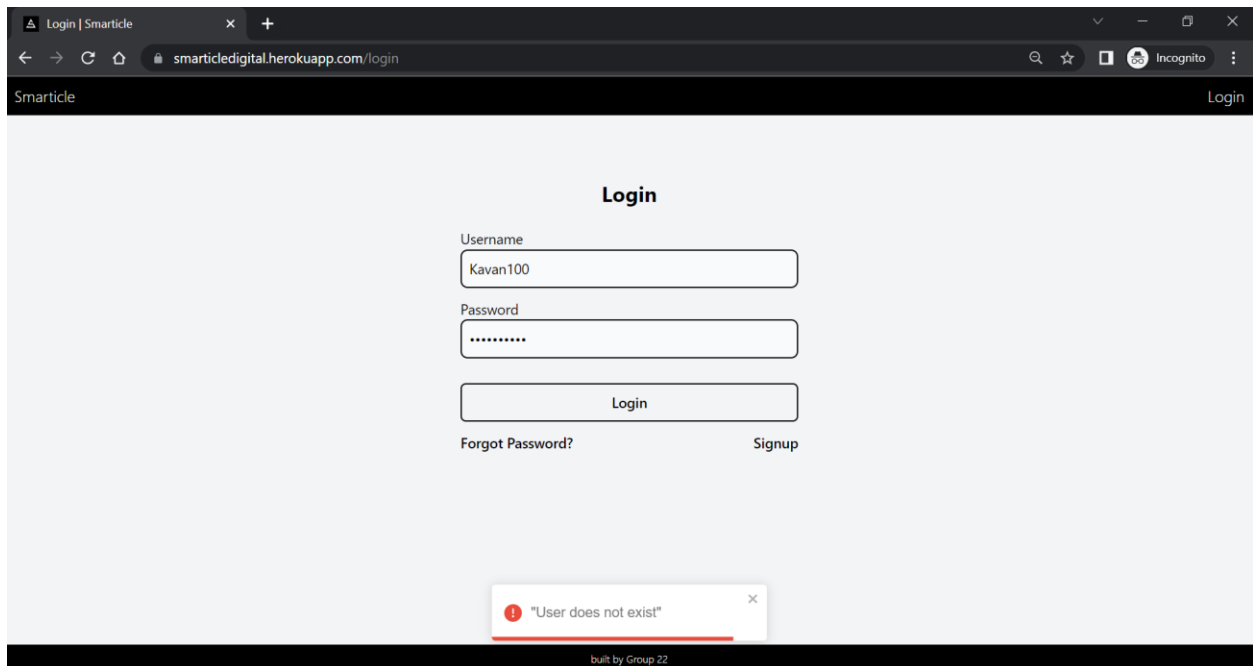
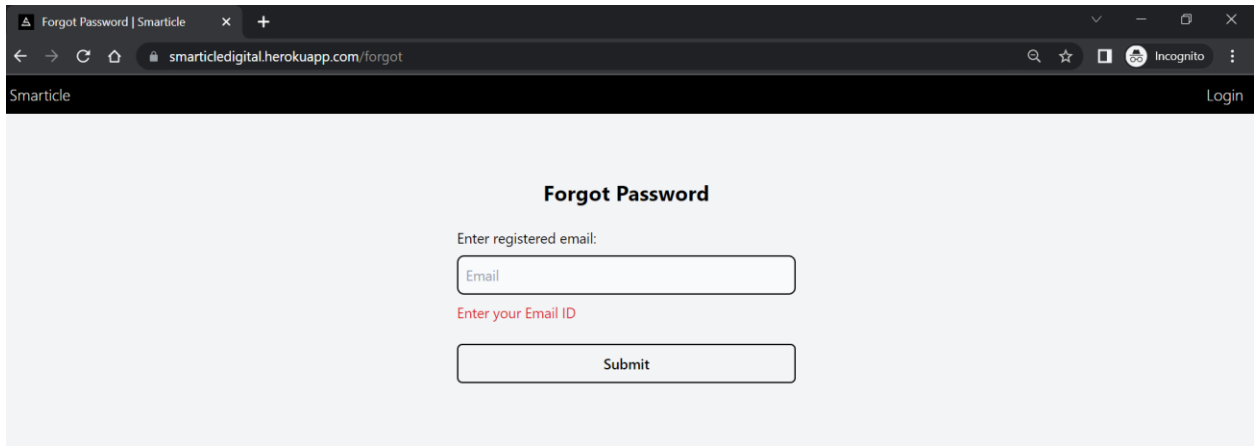


Figure 14: Invalid Username

5. Once registered, new users can log in to the application using their registration credentials. The user can use the forgot password link to reset their password if they no longer remember their

credentials. The user must provide their email address to reset their password. An invalid email id error will be displayed if the user gives the wrong e-mail address that was not used at the time of registration.



Forgot Password | Smarticle

smarticledigital.herokuapp.com/forgot

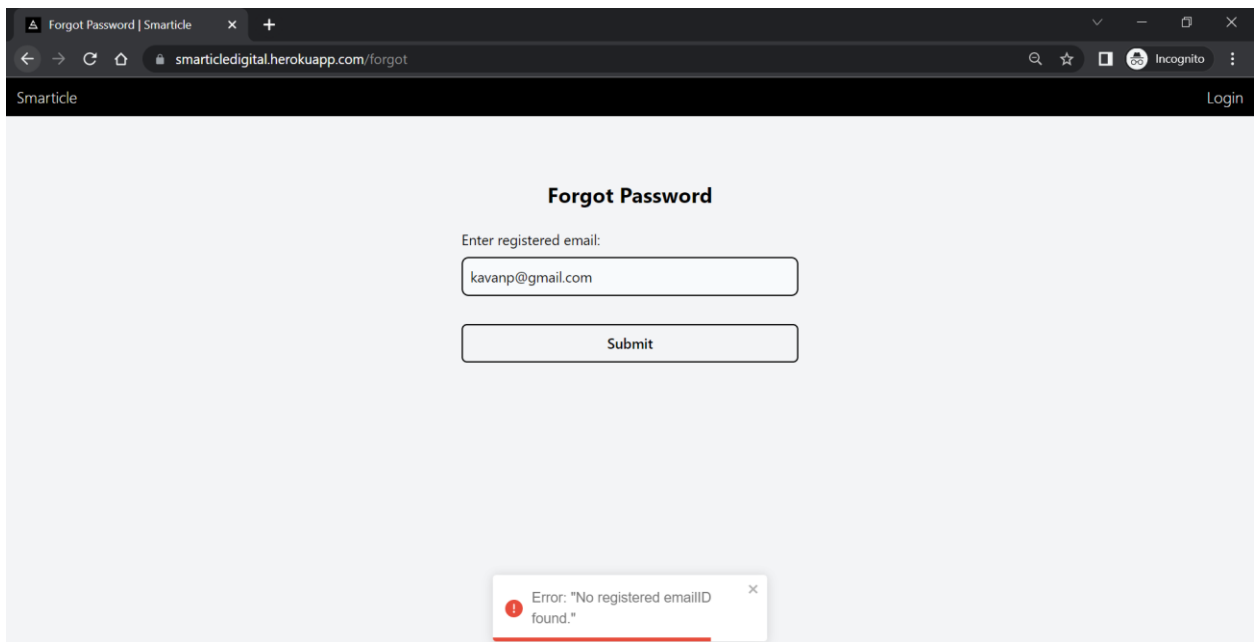
Smarticle Login

Forgot Password

Enter registered email:

Enter your Email ID

Figure 15: Throws error without entering email



Forgot Password | Smarticle

smarticledigital.herokuapp.com/forgot

Smarticle Login

Forgot Password

Enter registered email:

Error: "No registered emailID found."

Figure 16: Error while entering the wrong email

6. Forgot password

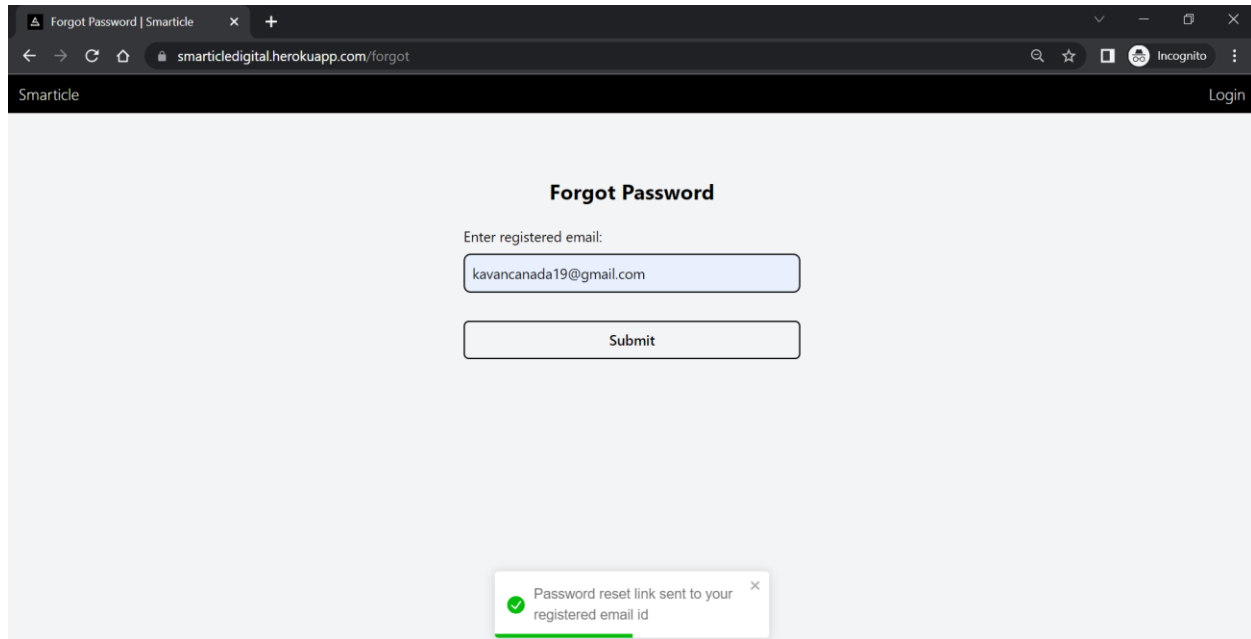


Figure 17: Forgot password successful

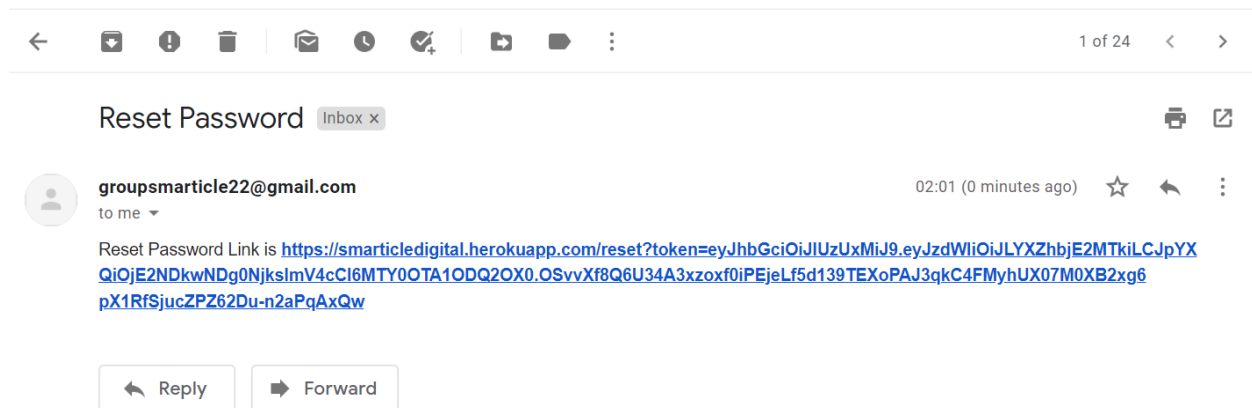


Figure 18: Reset password link sent to the registered email

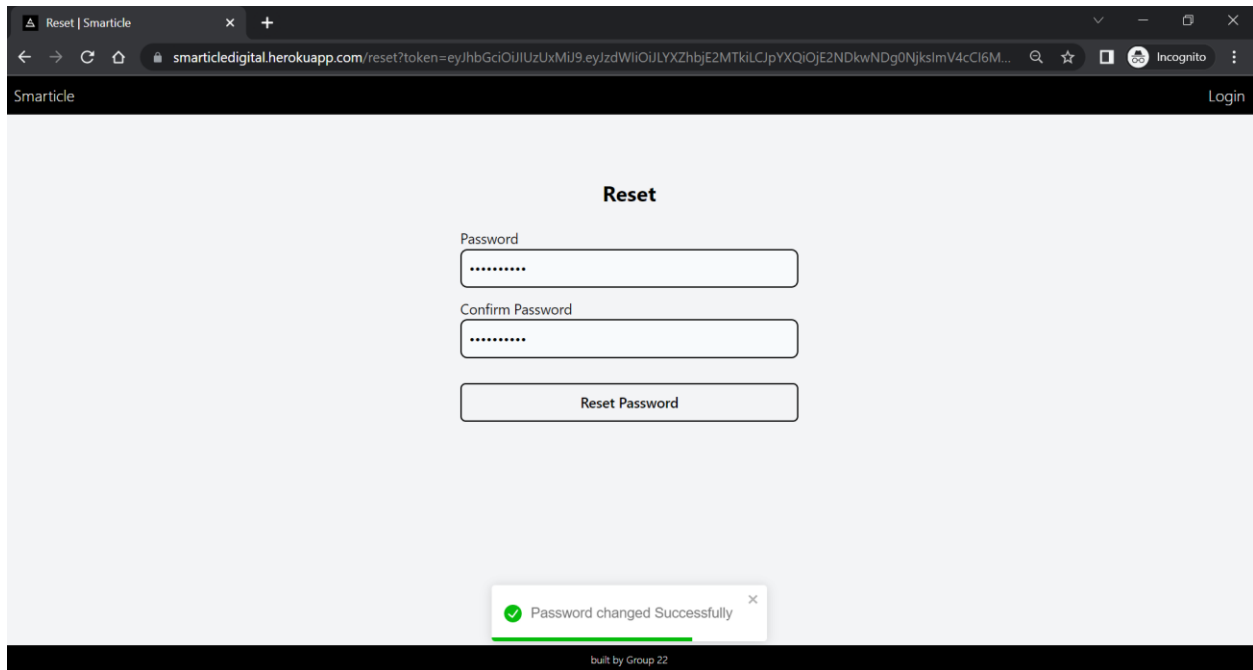


Figure 19: Reset password successful

- Users can only view public articles before logging in. Once logged in, users can access both public and private articles. If the user sets their preferences for article tags, then the user will see the articles related to their preferences.

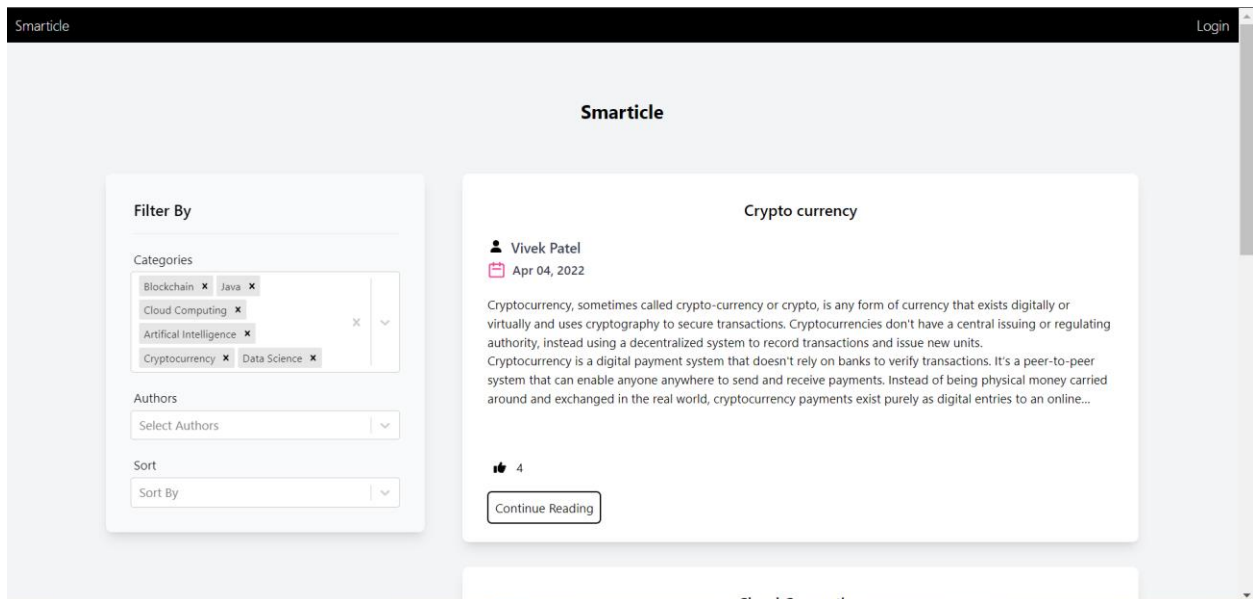


Figure 20: Public view of the article

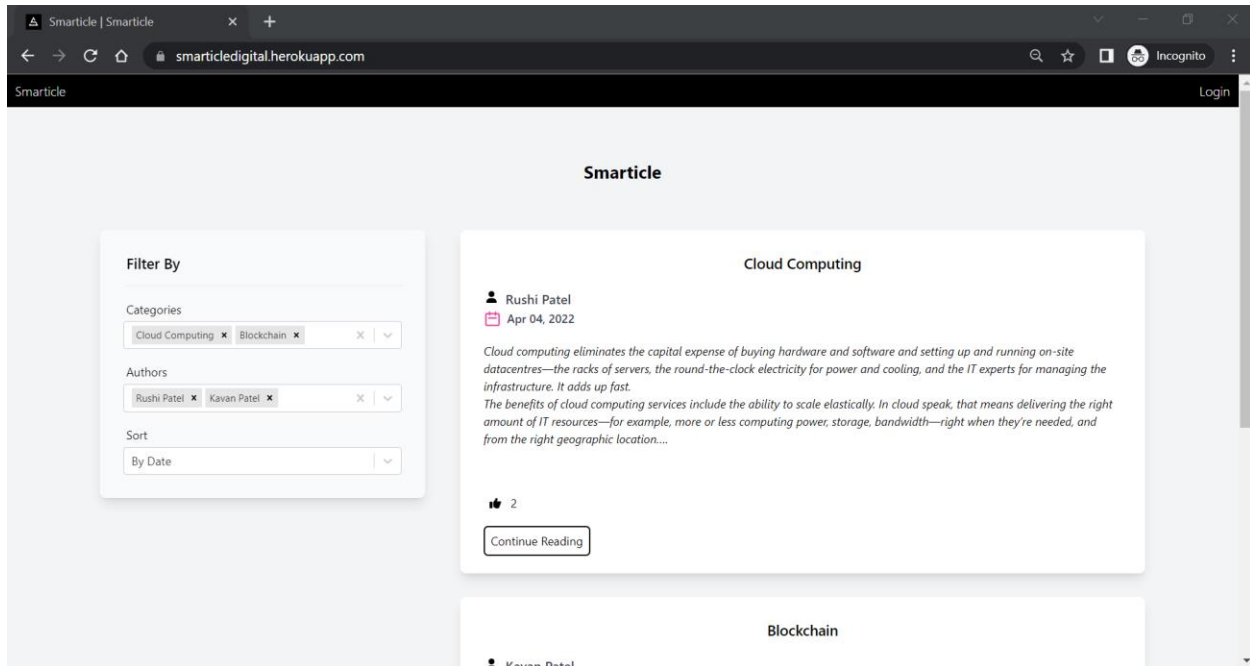


Figure 21: Filtering the article by date before login having tags cloud and blockchain for authors Rushi and Kavan

8. New users may post articles and apply bold, italic, bullets, and many other formatting widgets to their articles. Users can create a new tag by entering the tag name in the drop-down menu.

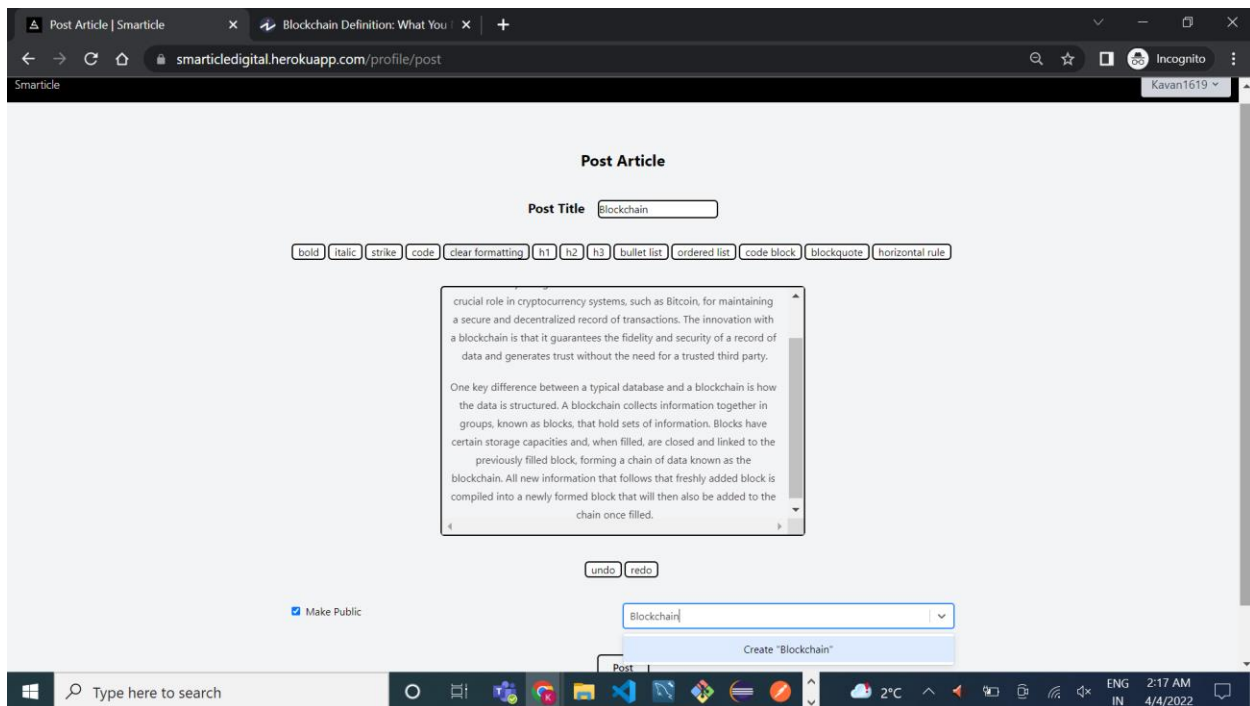


Figure 22: Creating a new tag Blockchain

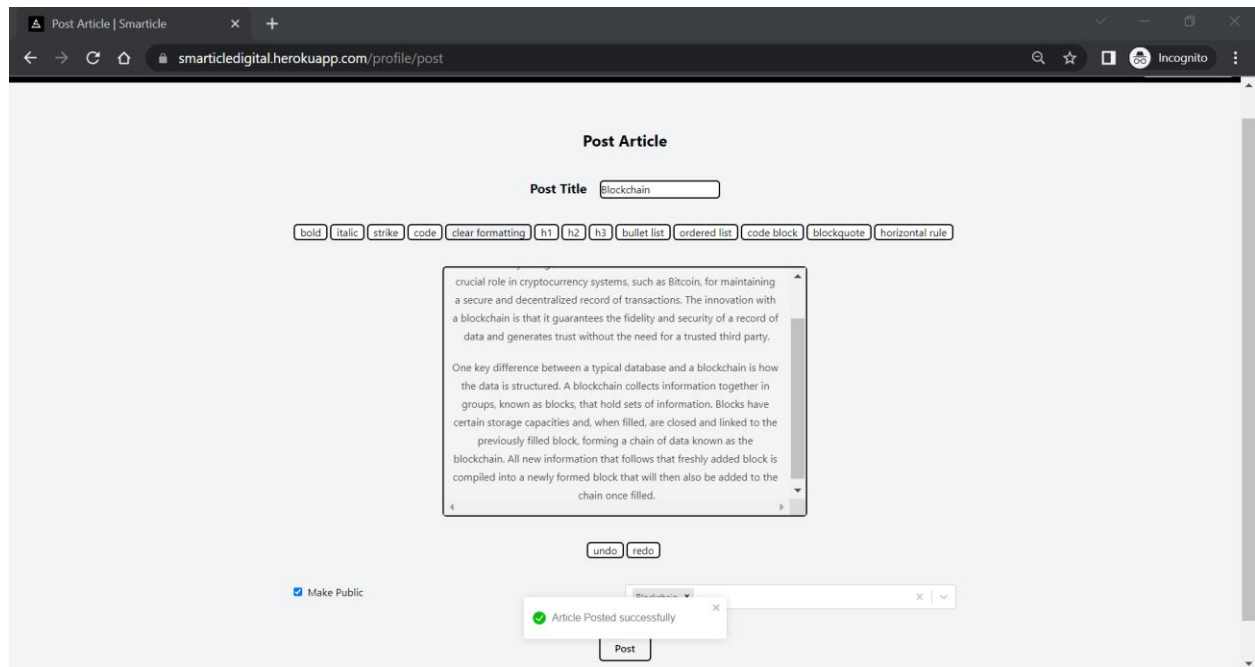


Figure 23: Article posted successfully

9. Tag preferences can be selected and removed from the preference management drop-down menu on the profile page.

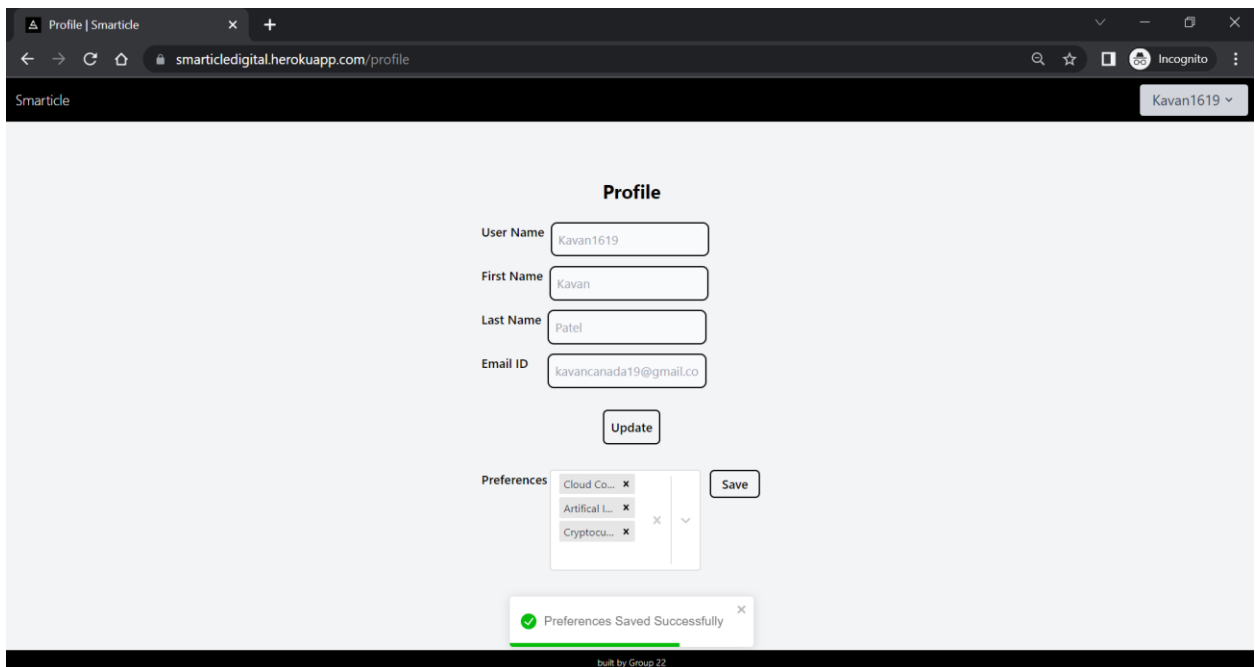


Figure 24: Saved Preference for Cloud, AI, and cryptocurrency

10. User can update their profile by clicking on the dropdown menu from the navbar section.

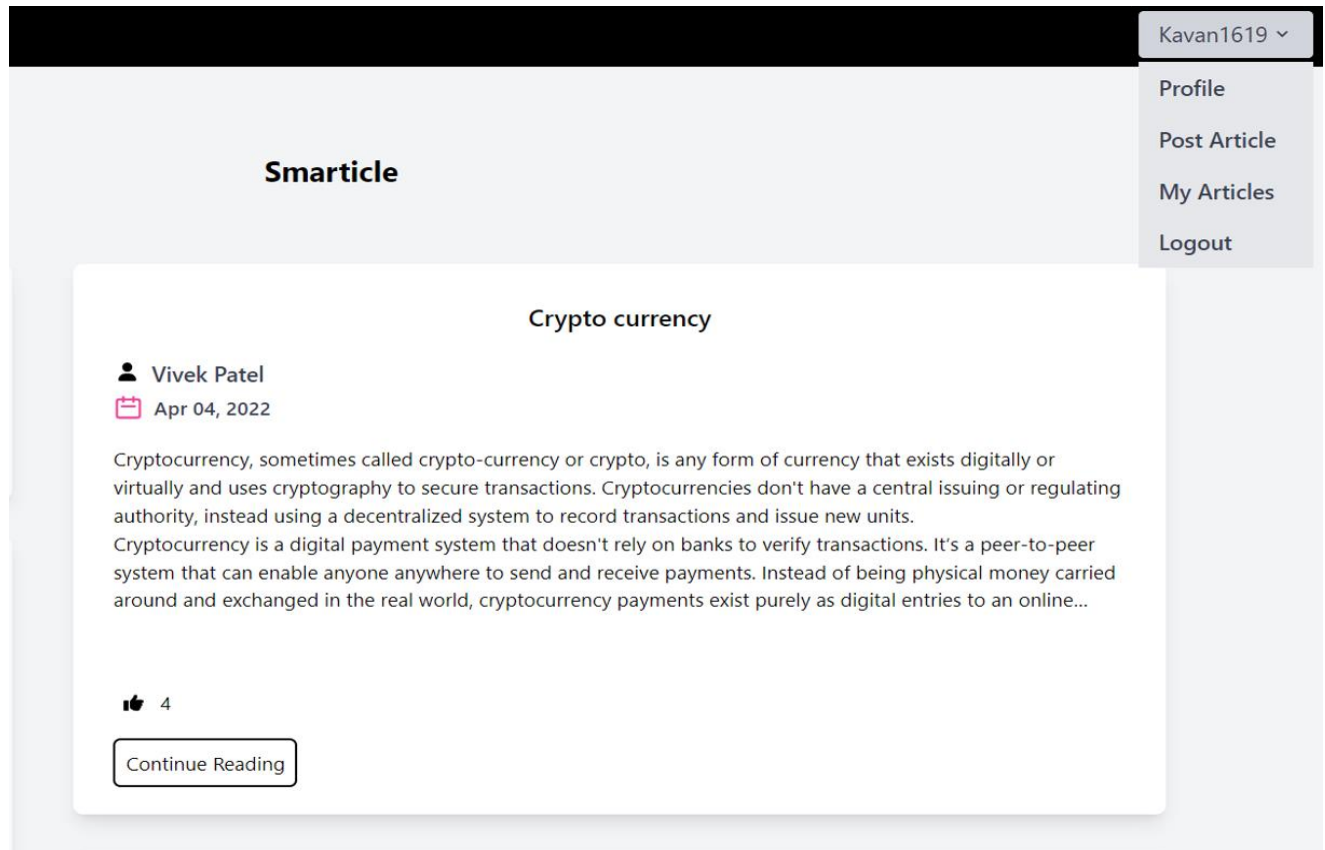


Figure 25: Drop-down menu

11. Users will be able to update their first name, last name, and email

The screenshot shows a web browser window with the address bar displaying 'smarticledigital.herokuapp.com/profile'. The page title is 'Profile | Smarticle'. The user's name 'Kavan1619' is shown in the top right corner. The main content area is titled 'Profile' and contains the following fields:

- User Name: Kavan1619
- First Name: Kavan
- Last Name: Patel
- Email ID: kavancanada19@gmail.co

Below these fields is an 'Update' button. Underneath the 'Update' button is a 'Preferences' section with a list of items: 'Artificial I...', 'Cloud Co...', and 'Cryptocu...'. Each item has a close button (X). To the right of the list is a 'Save' button.

Figure 26: Before updating the profile

The screenshot shows the same web browser window as Figure 26, but with the following updated information:

- User Name: Kavan1619
- First Name: KavanP
- Last Name: PatelK
- Email ID: kavancanada100@gmail.c

The 'Update' button is still present. The 'Preferences' section remains the same. A green notification banner at the bottom of the page reads 'Details Updated Successfully' with a close button (X).

Figure 27: Updating Firstname, Lastname, and email

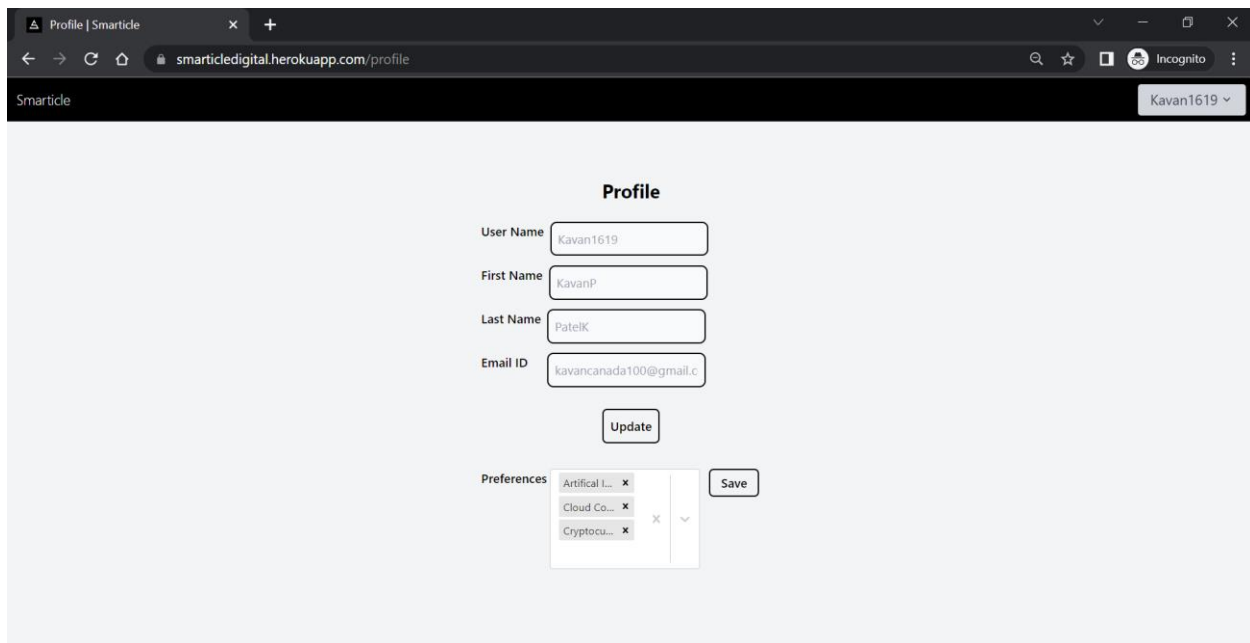


Figure 28: After Updating Profile

12. User can see their posted articles from the My articles dropdown menu.

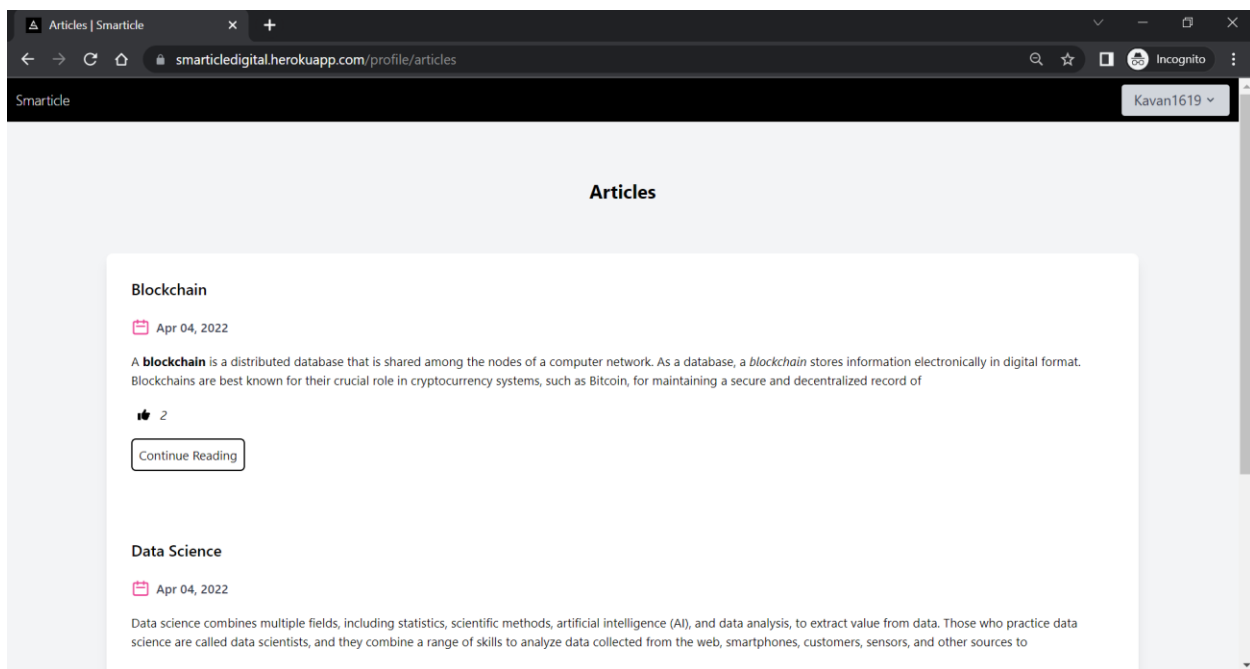


Figure 29: My Article page

13. The articles can be liked or disliked by users. After logging in, the like and dislike feature will be available.

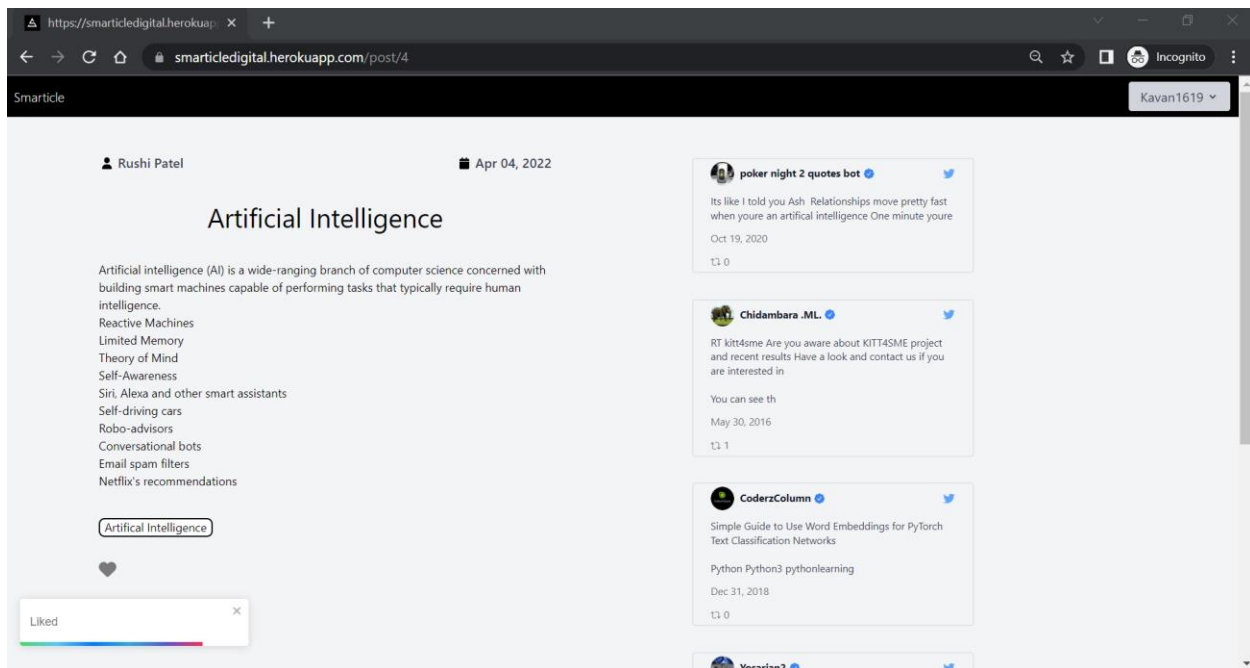


Figure 30: The user liked the article

14. Moreover, the user can also dislike the posted article.

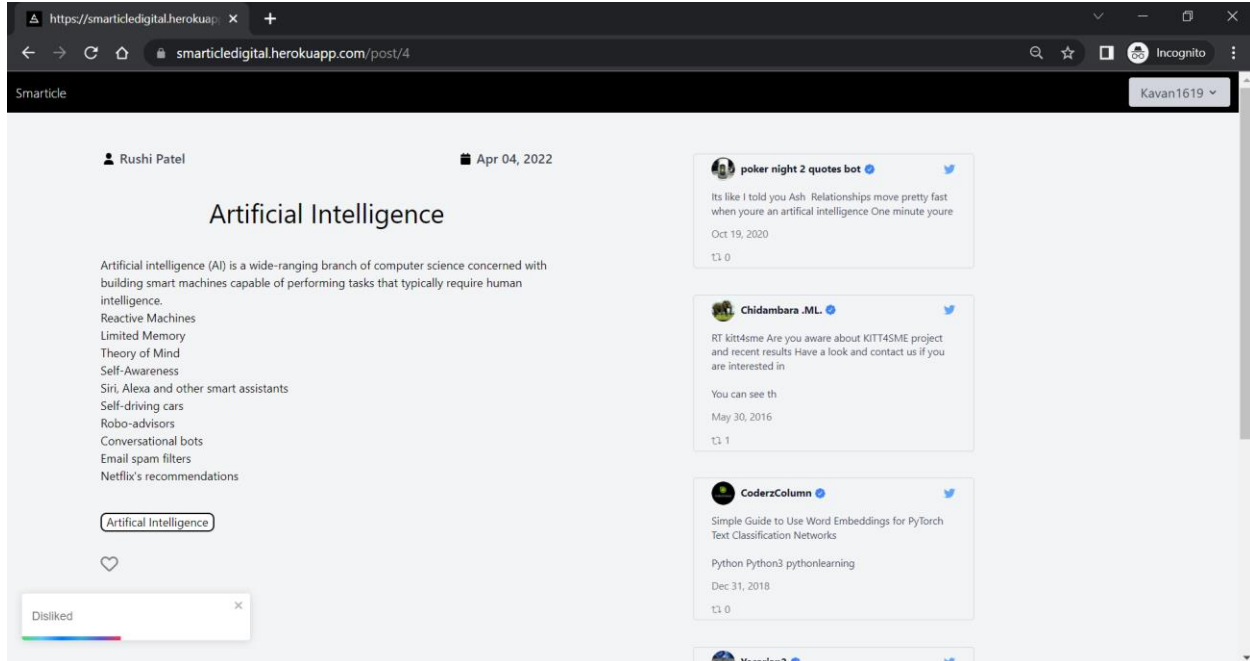


Figure 31: User disliked the article

15. Users can also view the like count.

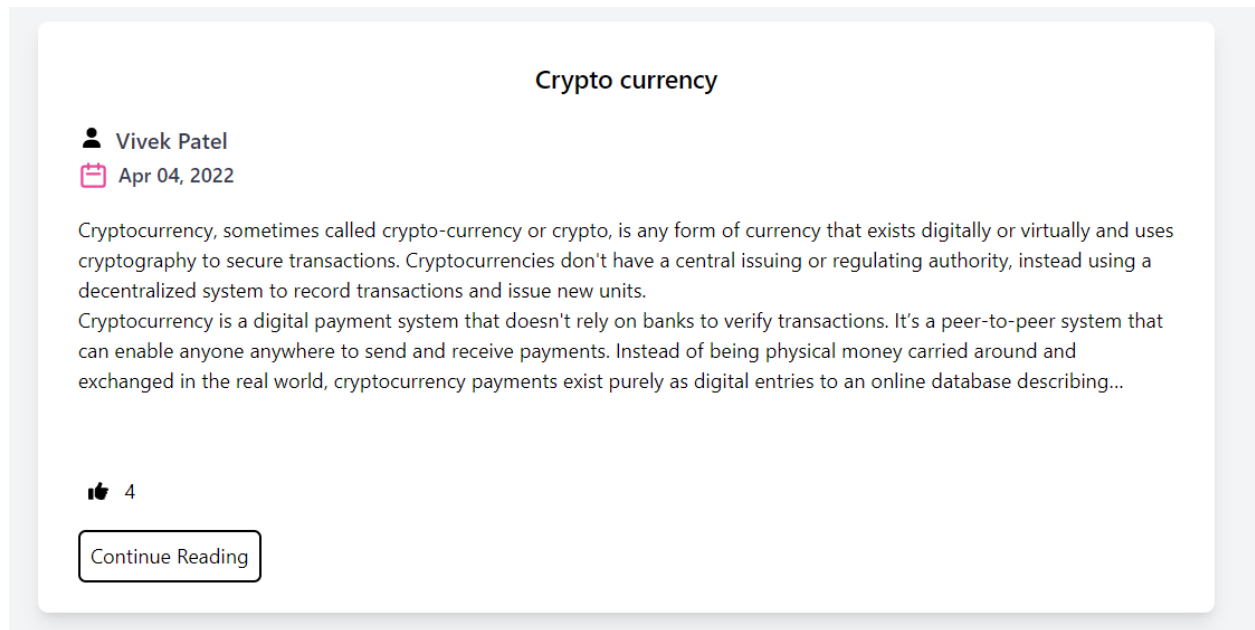


Figure 32: Like count visible for each article

16. When uploading an article, you can choose whether to make it public or private by ticking the Make public option at the bottom of the post article page. If this box is not ticked, the article will be published as private.

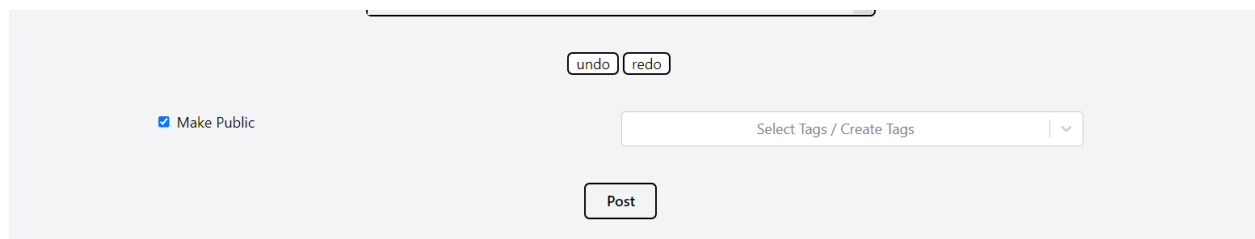


Figure 33: Post the article

17. Users can view the tweet count based on the tag preference on the home page as shown in the below figure. This feature is only visible to logged-in users.

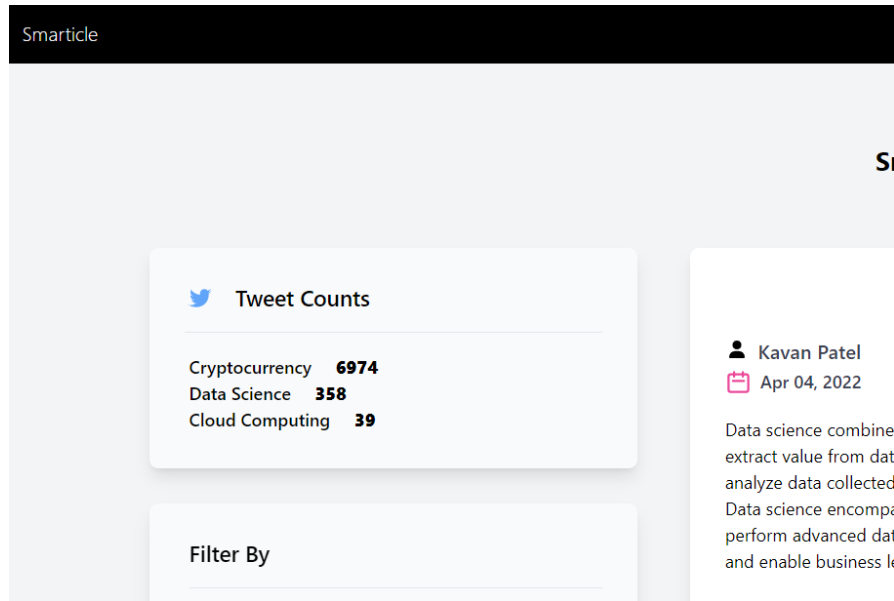


Figure 34: Tweet counts based on user preference

18. The advance filter feature option includes sorting the article based on date or like count. Moreover, the user can also filter the article based on the author's name and categories.

If a user specifies certain tags as their choice, they will be saved in the database. If the user signs in again, the tags that the user previously specified will be automatically set in the categories area, and articles based on the user's preferences will be fetched and presented until the user changes their preferences. If the user is not logged in, all public articles with all tags are automatically selected in the categories part of the filter menu.

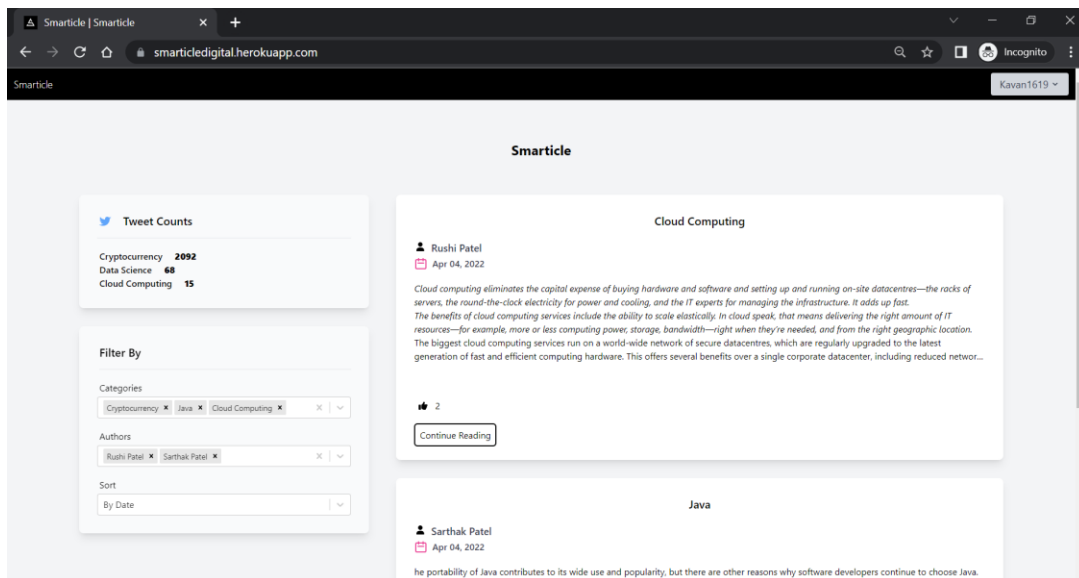


Figure 35: Advanced filter for articles by date having tags cryptocurrency, java, and cloud for authors Sarthak Patel and Rushi Patel

19. While reading the content, the top five tweets for a tag associated with the article will be fetched.

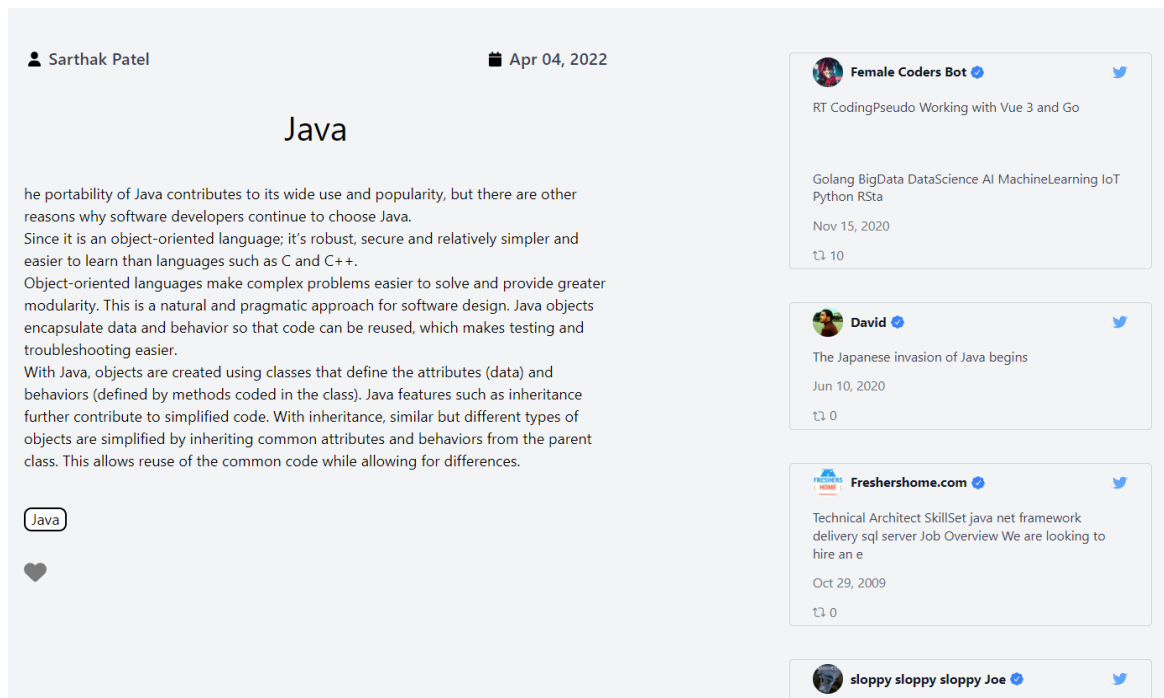


Figure 36: Showing the top 5 tweets for Tag Java

Use Case Diagram:

Smarticle Application

Use case diagram
of smarticle

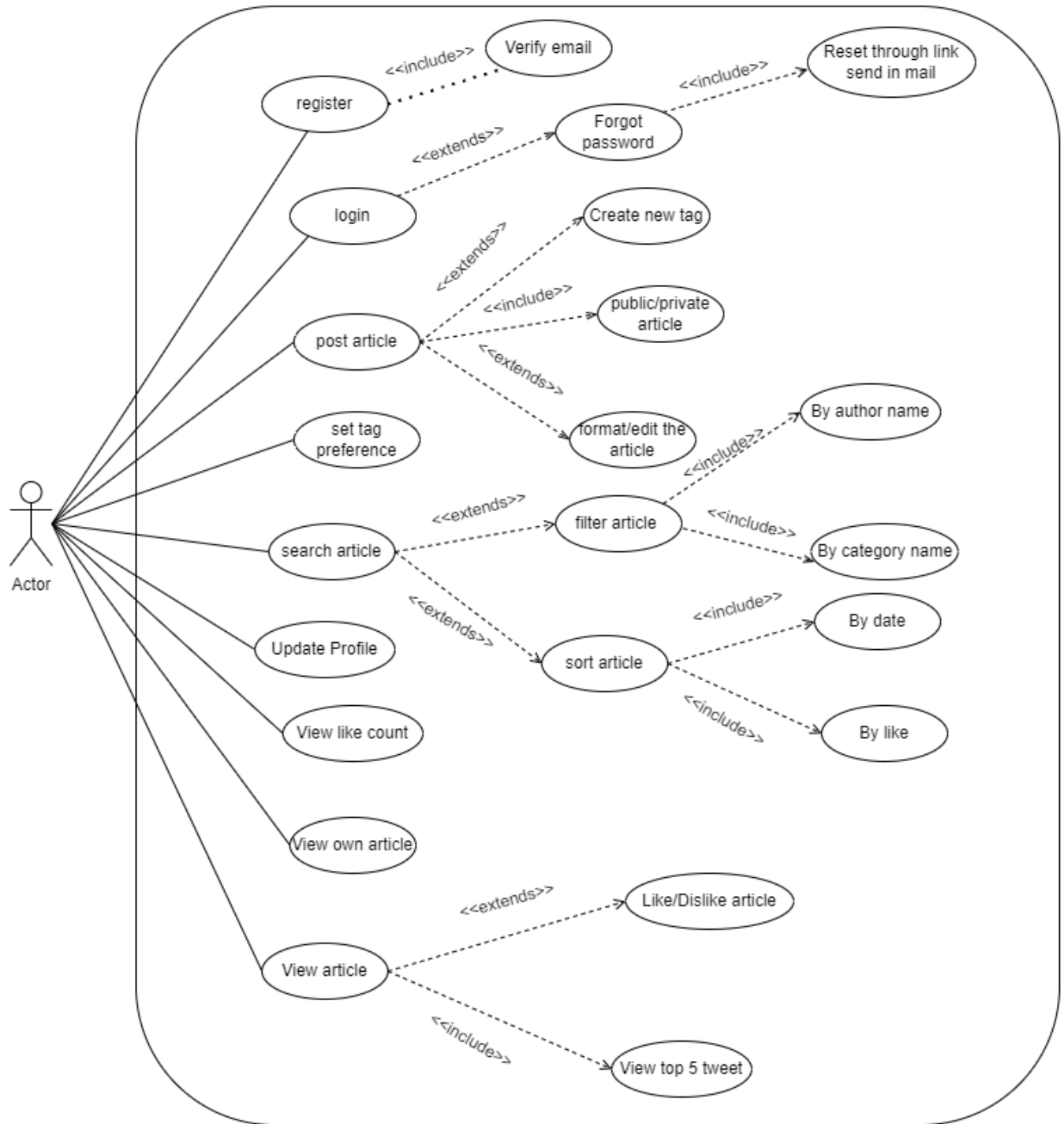


Figure 37: Use case diagram

ER Diagram

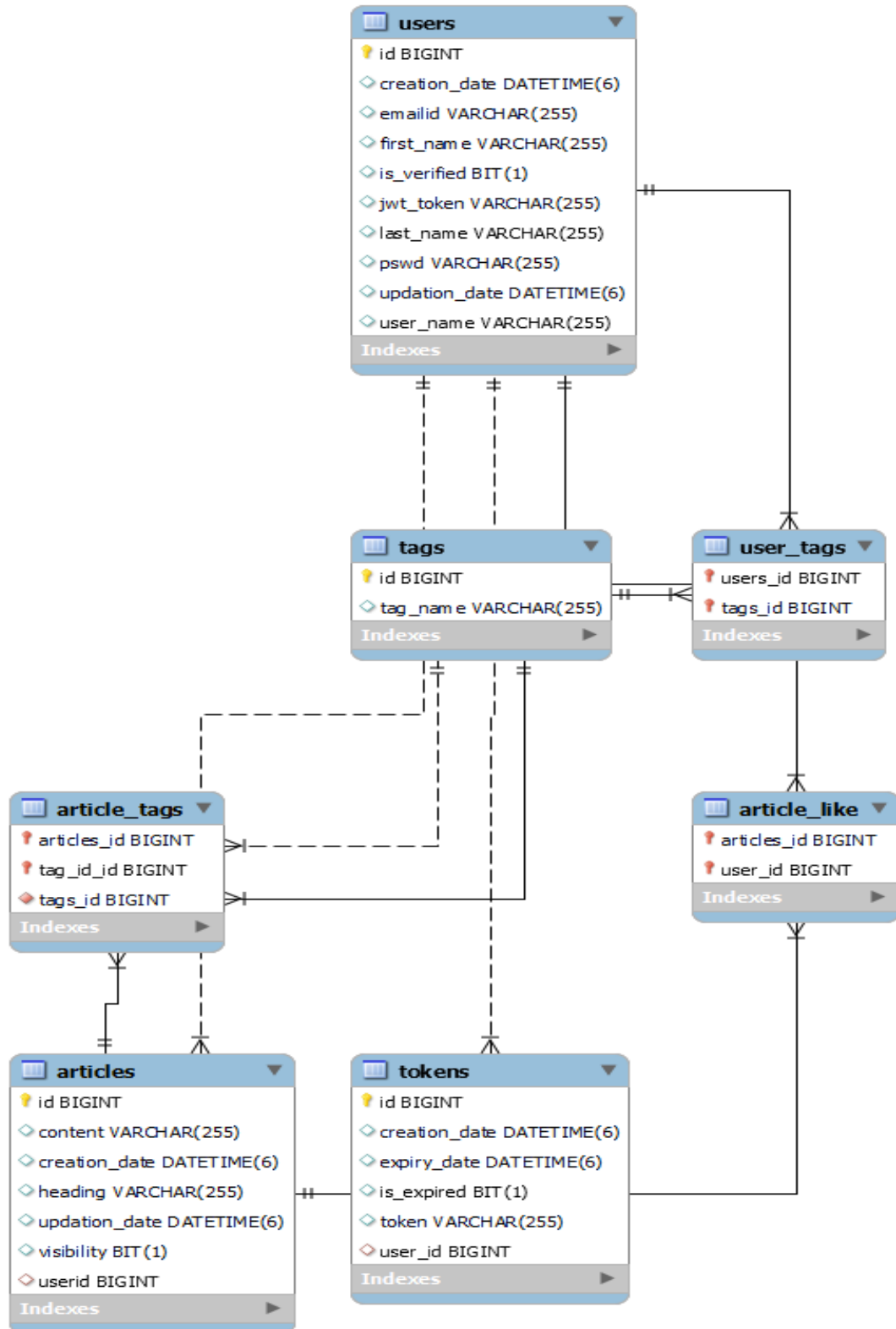


Figure 38: Smarticle Database ER Diagram

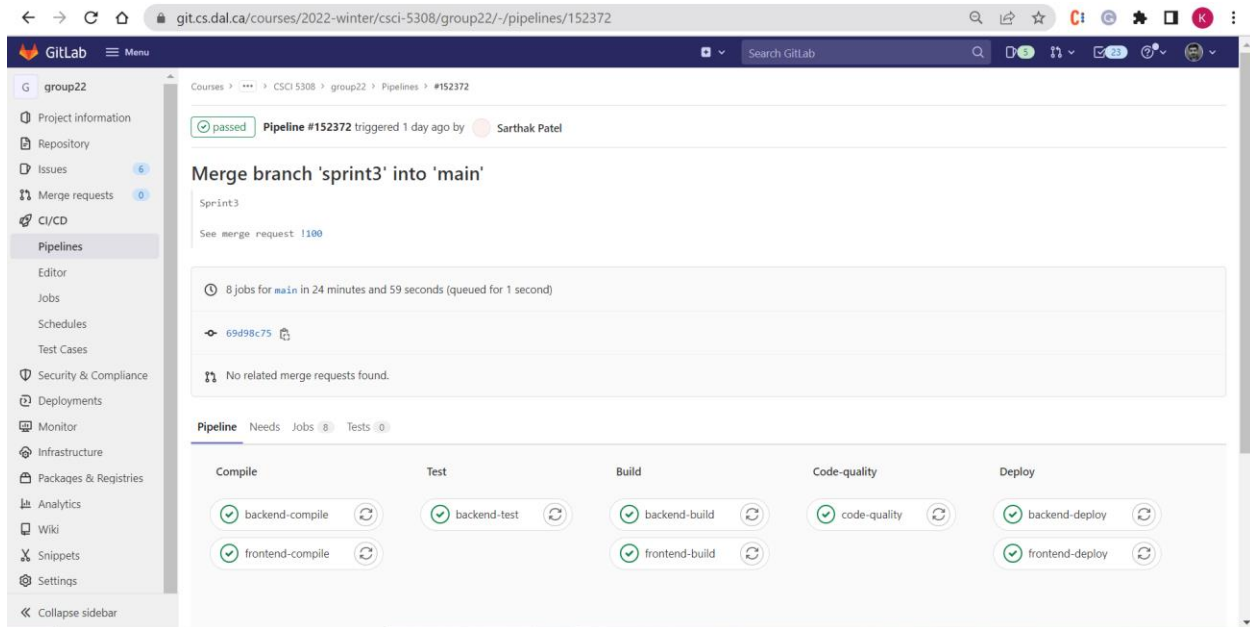


Figure 39: CI/CD Pipeline for both backend and frontend with Code Quality

```
Administrator: C:\Windows\System32\cmd.exe
-Total implementation smell instances detected-
  Abstract function call from constructor: 0      Complex conditional: 1
  Complex method: 0      Empty catch clause: 1
  Long identifier: 0      Long method: 0
  Long parameter list: 1  Long statement: 10
  Magic number: 3 Missing default: 0
----
Done.

H:\dalhousie\Term2\ASDC\Project\smell\backend>java -jar DesigniteJava.jar -i "H:\dalhousie\Term2\ASDC\Project\smell\backend" -o "H:\dalhousie\Term2\ASDC\Project\smell\backend" -f xml
Searching classpath folders ...
DesigniteJava Enterprise. Version 2.0.2
Copyright (C) 2022 Designite. All rights reserved.
Parsing the source code ...
Resolving symbols...
Computing metrics...
Detecting code smells...
Exporting analysis results...
Wrapping up ...
--Analysis summary--
  Total LOC analyzed: 1559      Number of packages: 14
  Number of classes: 43      Number of methods: 219
-Total architecture smell instances detected-
  Cyclic dependency: 18      God component: 0
  Ambiguous interface: 0      Feature concentration: 0
  Unstable dependency: 4      Scattered functionality: 1
  Dense structure: 0
-Total design smell instances detected-
  Imperative abstraction: 0      Multifaceted abstraction: 0
  Unnecessary abstraction: 1      Unutilized abstraction: 11
  Feature envy: 0      Deficient encapsulation: 2
  Unexploited encapsulation: 0      Broken modularization: 1
  Cyclically-dependent modularization: 0      Hub-like modularization: 0
  Insufficient modularization: 1      Broken hierarchy: 3
  Cyclic hierarchy: 0      Deep hierarchy: 0
  Missing hierarchy: 0      Multipath hierarchy: 0
  Rebellious hierarchy: 0      Wide hierarchy: 0
-Total testability smell instances detected-
  Hard-wired dependency: 5      Global state: 2
  Excessive dependency: 3      Law of Demeter violation: 4
-Total implementation smell instances detected-
  Abstract function call from constructor: 0      Complex conditional: 3
  Complex method: 0      Empty catch clause: 1
  Long identifier: 0      Long method: 0
  Long parameter list: 1      Long statement: 11
  Magic number: 6 Missing default: 0
----
Done.
```

Figure 40: Smells Detected

```

-Total implementation smell instances detected-
  Abstract function call from constructor: 0      Complex conditional: 0
  Complex method: 0      Empty catch clause: 0
  Long identifier: 0      Long method: 0
  Long parameter list: 1  Long statement: 49
  Magic number: 0 Missing default: 0
----
Done.
H:\dalhousie\Term2\ASDC\Project\group22\backend>java -jar DesigniteJava.jar -i "H:\dalhousie\Term2\ASDC\Project\group22\backend" -o "H:\dalhousie\Term2\ASDC\Project\group22\backend\output"
Searching classpath folders ...
DesigniteJava Enterprise, Version 2.0.2
Copyright (C) 2022 Designite. All rights reserved.
Parsing the source code ...
Resolving symbols...
Computing metrics...
Detecting code smells...
Exporting analysis results...
Wrapping up ...
--Analysis summary--
  Total LOC analyzed: 2972      Number of packages: 14
  Number of classes: 66      Number of methods: 350
-Total architecture smell instances detected-
  Cyclic dependency: 20      God component: 0
  Ambiguous interface: 0      Feature concentration: 1
  Unstable dependency: 5      Scattered functionality: 8
  Dense structure: 1
-Total design smell instances detected-
  Imperative abstraction: 0      Multifaceted abstraction: 0
  Unnecessary abstraction: 1      Unutilized abstraction: 10
  Feature envy: 7      Deficient encapsulation: 3
  Unexploited encapsulation: 0      Broken modularization: 2
  Cyclically-dependent modularization: 0      Hub-like modularization: 0
  Insufficient modularization: 2      Broken hierarchy: 4
  Cyclic hierarchy: 0      Deep hierarchy: 0
  Missing hierarchy: 0      Multipath hierarchy: 0
  Rebellious hierarchy: 0      Wide hierarchy: 0
-Total testability smell instances detected-
  Hard-wired dependency: 4      Global state: 3
  Excessive dependency: 6      Law of Demeter violation: 9
-Total implementation smell instances detected-
  Abstract function call from constructor: 0      Complex conditional: 0
  Complex method: 0      Empty catch clause: 0
  Long identifier: 0      Long method: 0
  Long parameter list: 1      Long statement: 49
  Magic number: 0      Missing default: 0
----
Done.
H:\dalhousie\Term2\ASDC\Project\group22\backend>

```

Figure 41: After refactoring

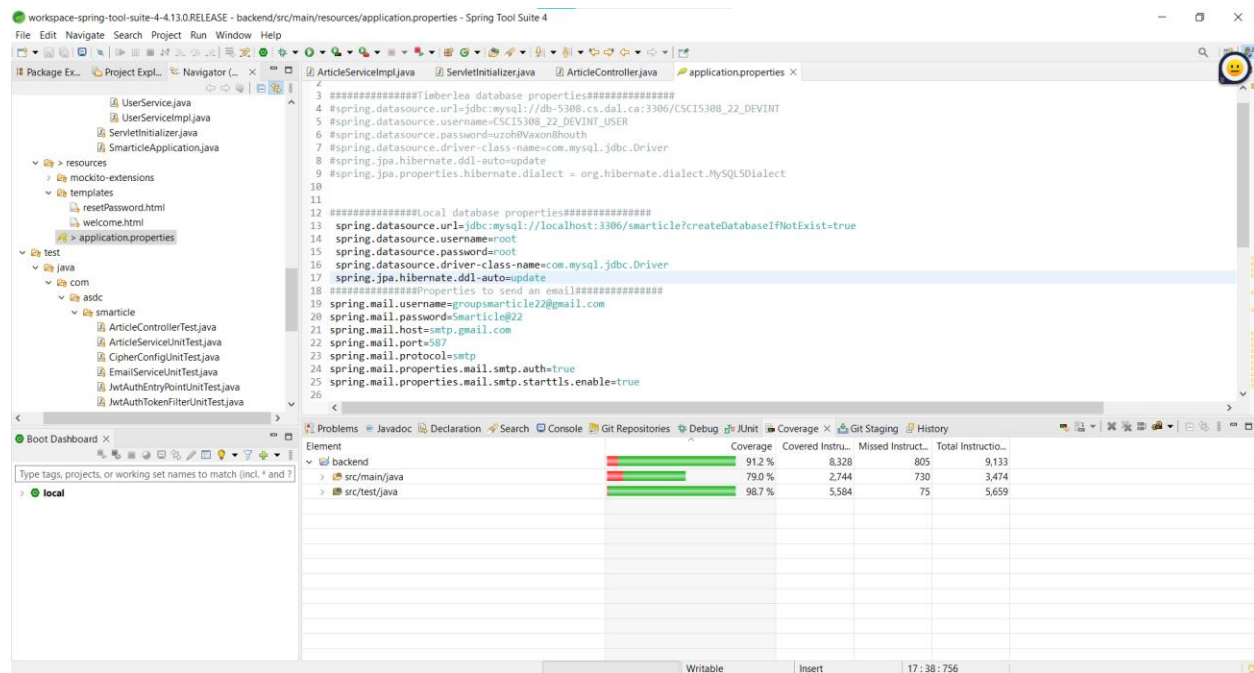


Figure 42: Code test coverage

List of Smells unable to remove with justification.

1. Long parameter list:

This implementation smell occurs due to in built constructor of class UserDetailsImpl of the spring security. It is not possible to change inbuilt constructor.

2. Long statement:

This implementation smell occurs for usage creating mock HTTP requests, JPA methods to filter and sort data, and spring security methods for resolving API requests.

3. Unnecessary abstraction:

This smell occurs because we have declared all applications constant in a single java file without having any methods according to standard practice but in designation is considered a smell.

4. Unutilized abstraction:

This smell occurs because of classes such as controller, security, and email service. In spring, these classes are directly used by REST API to process the request. Hence, It is not possible to refactor this smell.

5. Feature envy:

This smell occurs because FilterPojo is used in the article service test class. It is not possible to write test code inside FilterPojo.Hence unable to remove the smell.

6. Deficient encapsulation:

This smell occurs because of all classes containing application constants such as rest api URL, error or message, and magic constant. According to development practice, all constants are declared as static final inside a class. Thus unable to remove the smell.

7. Broken hierarchy:

We have created a common class to represent the structure of the http response and extend that class in each controller to have common response structure but in designate it is considered as smell as we have not override it.Thus unable to remove the smell.

8. Insufficient modularization

This smell occurs because of public methods inside the entity class. Usually, these methods are getter and setter methods. Thus unable to remove the smell.

9. Cyclic dependency

We have used the concept of cardinality to define the relationship between entities using JPA. We have used one too many, many to one but in designate, it is considered a cyclic dependency. It is not possible to write code without mapping in JPA. Hence unable to remove the smell.

10. Unstable dependency

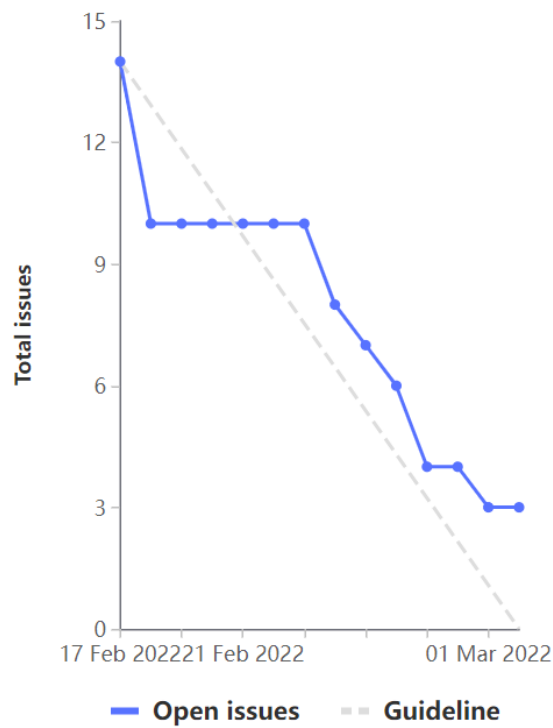
According to spring framework norms service layer is autowired and used in other classes and we have followed similar practice but in designation, it is considered as an unstable dependency as service class used more in other classes which is not according to spring framework norms.

11. Dense structure

This smell occurs due to a large number of import statements. It is necessary to import all API classes for development. Thus unable to remove the smell.

Progress chart

Burndown chart



Burnup chart

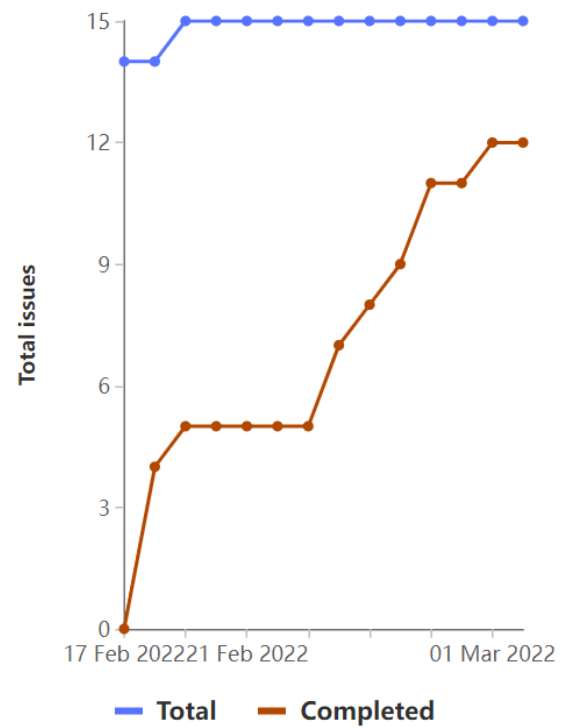
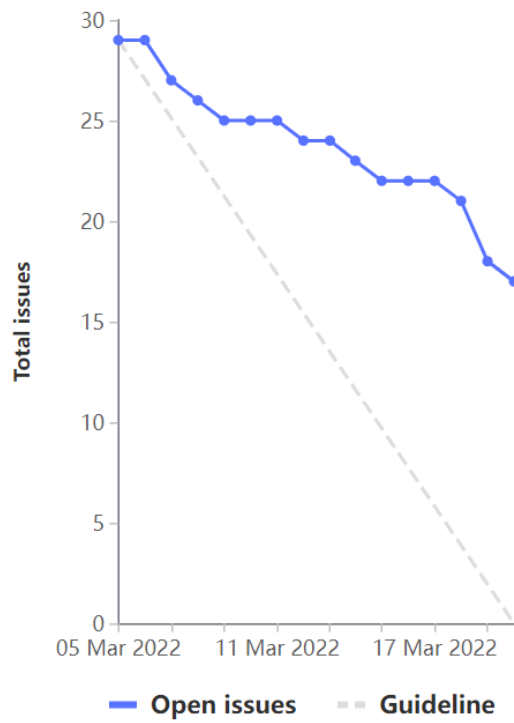


Figure 42: Sprint 1

Burndown chart



Burnup chart

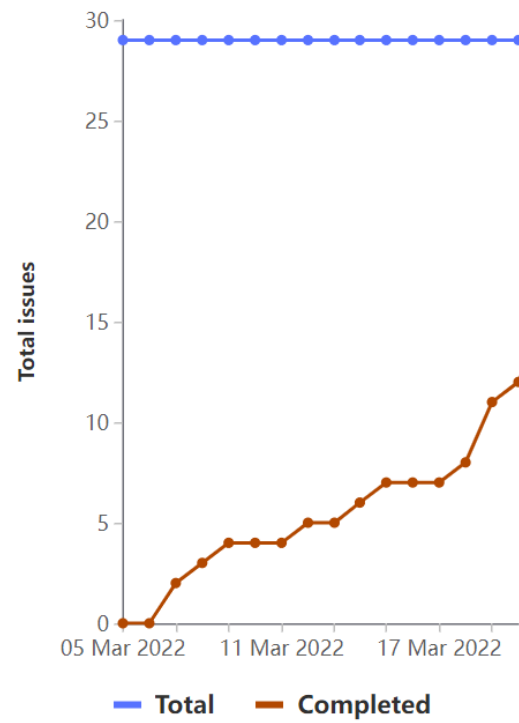
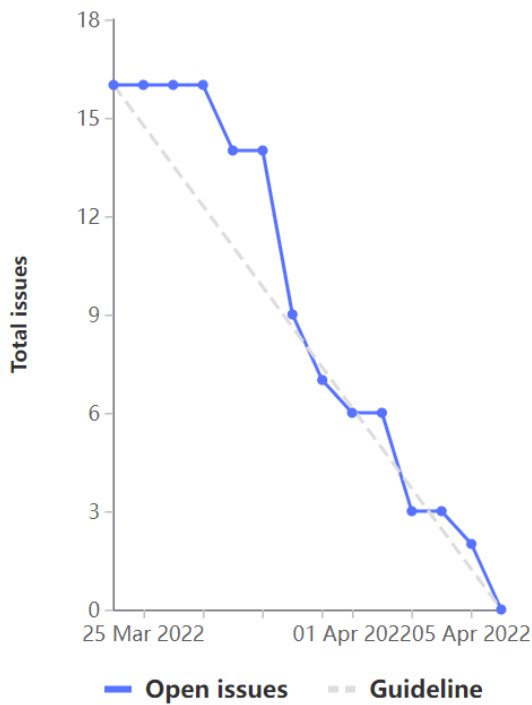


Figure 43: Sprint 2

Burndown chart



Burnup chart

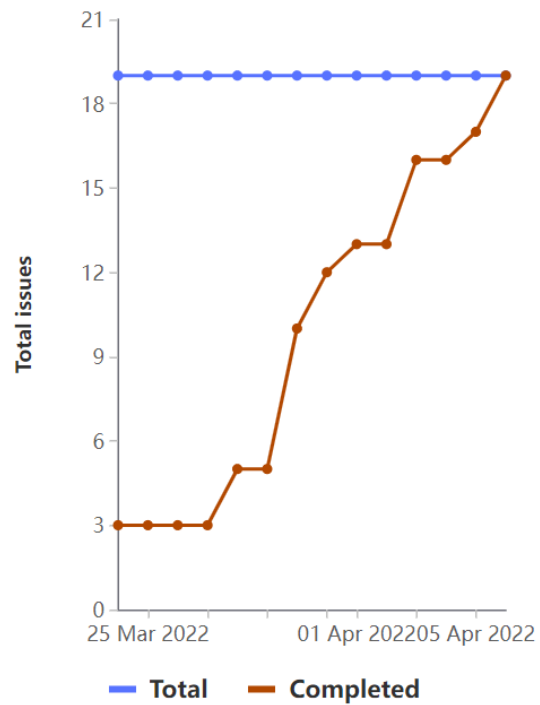


Figure 44: Sprint 3

Feature contribution

The following tasks were completed by all members of the group, and they all contributed the same amount.

- Test Case Refactoring
- Code Refactoring
- Analysis for Documentation
- Documentation
- Code Cleaning
- Removing Code Smells

Feature Number	Features	Contributor Name	Contribution
1	UI design for the login module.	Kavan Patel	100%

2	Backend API for login module.	Sarthak Patel	100%
3	Integrated backend API with UI for login module.	Angela Gilhotra	100%
4	UI design for the registration module.	Kavan Patel	100%
5	Backend API for registration module.	Vivek Patel	100%
6	Integrated the registration backend API with the registration UI.	Angela Gilhotra	100%
7	Database Design	Rushi Patel, Vivek Patel, Sarthak Patel	Rushi Patel – 25% Vivek Patel – 40% Sarthak Patel – 35%
8	Backend Architecture	Khushboo Patel, Rushi Patel, Vivek Patel, Sarthak Patel	Khushboo Patel- 25% Rushi Patel – 25% Vivek Patel – 25% Sarthak Patel – 25%
9	CI Pipeline	Vivek Patel	100%
10	CD Pipeline	Vivek Patel	100%
10	UI design for Home Page	Angela Gilhotra	100%
11	UI design for home page - public search view before login	Kavan Patel	100%
12	Integrated UI with backend API of Homepage public view before login	Angela Gilhotra	100%
13	Backend API of Home Page (public view before login)	Sarthak Patel	100%

14	Backend API for the Tag count of the topics followed by the user	Khushboo Patel	100%
15	UI of tag counts followed by users	Khushboo Patel	100%
16	Integrated Backend API with UI for Tag count of the topics followed by the user	Vivek Patel	100%
17	Backend API for the post-article module	Khushboo Patel, Rushi Patel	Khushboo Patel-60% Rushi Patel – 40%
18	UI for the post-article module.	Angela Gilhotra	100%
19	Integrated the backend API and UI of the post-article module.	Kavan Patel	100%
20	UI for private search view after Login	Kavan Patel	100%
21	Backend API for private search view after Login	Sarthak Patel	100%
22	Integrated UI with Backend API for private search view after Login	Angela Gilhotra	100%
23	Backend API for view article details page.	Sarthak Patel	100%
24	Integrated UI with Backend API for view article details page	Sarthak Patel	100%
25	UI for view article details Page	Angela Gilhotra	100%
26	UI for my posted articles page	Sarthak Patel	100%
27	Backend API for my posted articles page	Kavan Patel	100%

28	Integrated UI with backend API for my posted articles page	Rushi Patel	100%
29	UI for user profile module	Vivek Patel, Sarthak Patel	Sarthak- 50% Vivek – 50%
30	Backend API for preference management module	Vivek Patel, Rushi Patel, Sarthak Patel	Vivek – 33% Rushi – 33% Sarthak – 33%
31	Backend API for profile management	Vivek Patel	100%
32	Integrated backend and UI of user profile and preference management	Vivek Patel	100%
33	UI for Filter module	Rushi Patel	100%
34	Backend API for a filter module	Angela Gilhotra, Kavan Patel	Angela - 50% Kavan – 50%
35	Integrated Backend API for Implementing filter section with UI.	Angela Gilhotra, Vivek Patel	Angela – 50% Vivek – 50%
36	Backend API for sorting in search functionality	Rushi Patel	100%
37	Sorting module UI	Khushboo Patel	100%
38	Integrated Backend API with UI for sorting in search functionality	Khushboo Patel	100%
39	Backend API for like feature	Khushboo Patel	100%
40	UI for like feature	Vivek Patel	100%
41	Integrated UI with backend API for like feature.	Rushi Patel	100%

42	Backend API for twitter details	Kavan Patel	100%
43	UI for twitter details	Kavan Patel	100%
44	Integrated UI and Backend API for fetching twitter details	Kavan Patel	100%
45	CI Quality Analysis check	Sarthak Patel	100%
46	Use Case Scenario	Angela Gilhotra, Khushboo Patel	Angela – 50% Khushboo – 50%
47	Backend API for adding new tags	Angela Gilhotra	100%
48	UI for adding new tags	Rushi Patel	100%
49	Presentation preparation	Rushi Patel, Kavan Patel	Rushi – 50% Kavan – 50%

Team Member	Total Tasks Completed
Angela Gilhotra	17
Khushboo Patel	14
Kavan Patel	17
Rushi Patel	16
Sarthak Patel	17
Vivek Patel	18

References

- [1] bezkoder, “Spring Boot Token based Authentication with Spring Security & JWT - BezKoder,” BezKoder, Oct. 15, 2019. [Online]. Available: <https://www.bezkoder.com/spring-boot-jwt-authentication/>. [Accessed: Apr. 06, 2022]
- [2] Ioram Gordadze, “Spring Security with JWT for REST API,” Toptal Engineering Blog, 2020. [Online]. Available: <https://www.toptal.com/spring/spring-security-tutorial>. [Accessed: Apr. 06, 2022].
- [3] S. Das, “Implementing JSON Web Token (JWT) Authentication using Spring Security | A Detailed Walkthrough,” Medium, Dec. 23, 2021. [Online]. Available: <https://medium.com/geekculture/implementing-json-web-token-jwt-authentication-using-spring-security-detailed-walkthrough-1ac480a8d970>. [Accessed: Apr. 06, 2022].
- [4] “Spring Boot Security + JWT Hello World Example | JavaInUse,” Javainuse.com, 2014. <https://www.javainuse.com/spring/boot-jwt> (accessed Apr. 06, 2022).