BLOGGING TOOL DEVELOPMENT WITH EXPRESS.JS AND SQLITE

Contents

Three-tier architecture

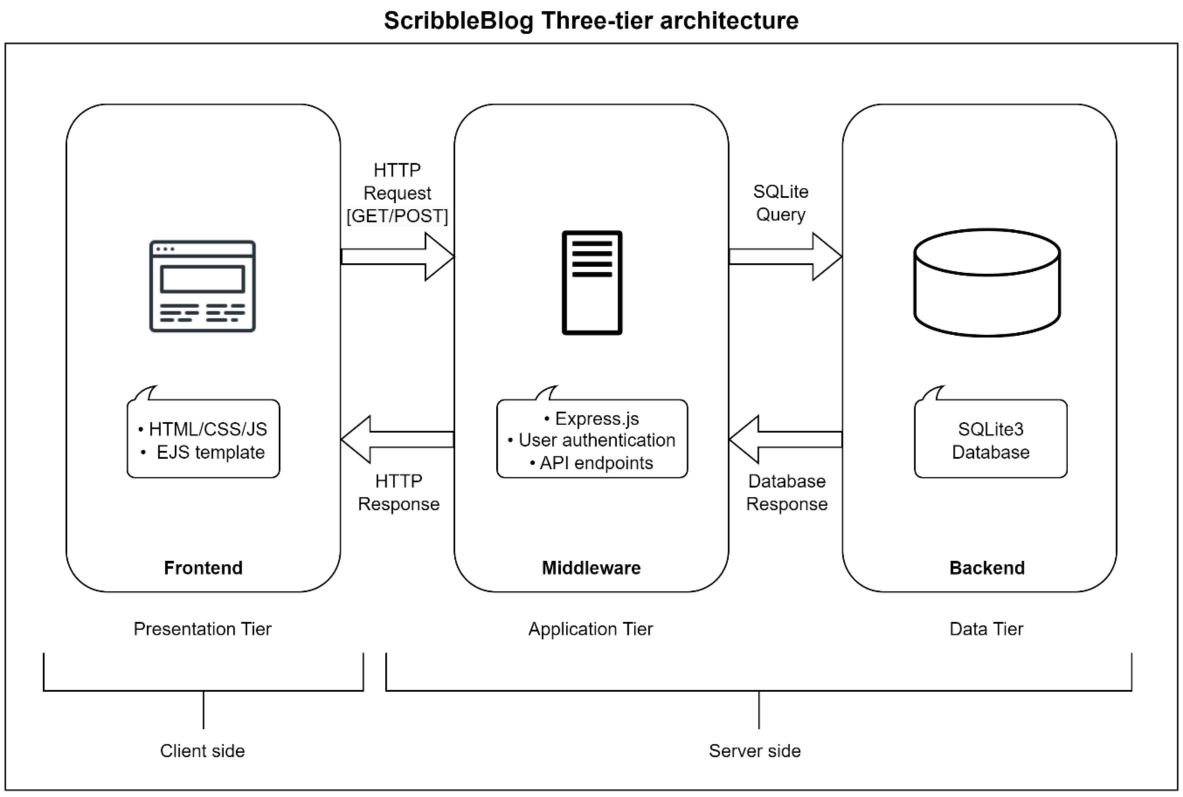


Figure 1. Schematic diagram

Figure 1 shows a high- level schematic diagram for my website—ScribbleBlog—three-tier architecture.

Frontend

HTML, CSS, JavaScript and EJS templates are elements representing the client-side functionality and user-interface. They oversee handling user interactions, rendering web pages and sending out HTTP requests such as GET and POST methods to the middleware.

Middleware

Express.js manages the routing and execution of the HTTP requests and responses. User authentication on the other hand, handles login credentials and session management to ensure that only authorized users are allowed access to the certain routes. API endpoints are used for different functionalities in the application such as fetching draft and published articles from the database and sending it back to the frontend. Other functionalities that use API endpoints include creating, editing, and deleting articles based on the specific article ID.

Backend

The applications data, including user information, data related to articles and comments are stored in the SQLite3 database. Depending on the requests from the middleware, the database is responsible for carrying out the database CRUD operations.

Database schema

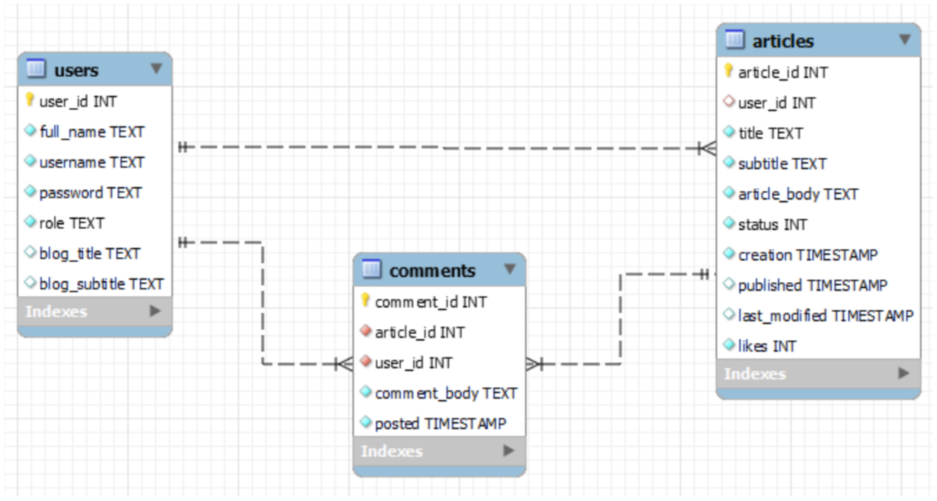


Figure 1. Database Schema of ScribbleBlog

Table	**Purpose**
users	Store information of registered users
articles	Stores data related to draft and published articles
comments	Stores comments posted by readers on individual articles

Page flow diagram

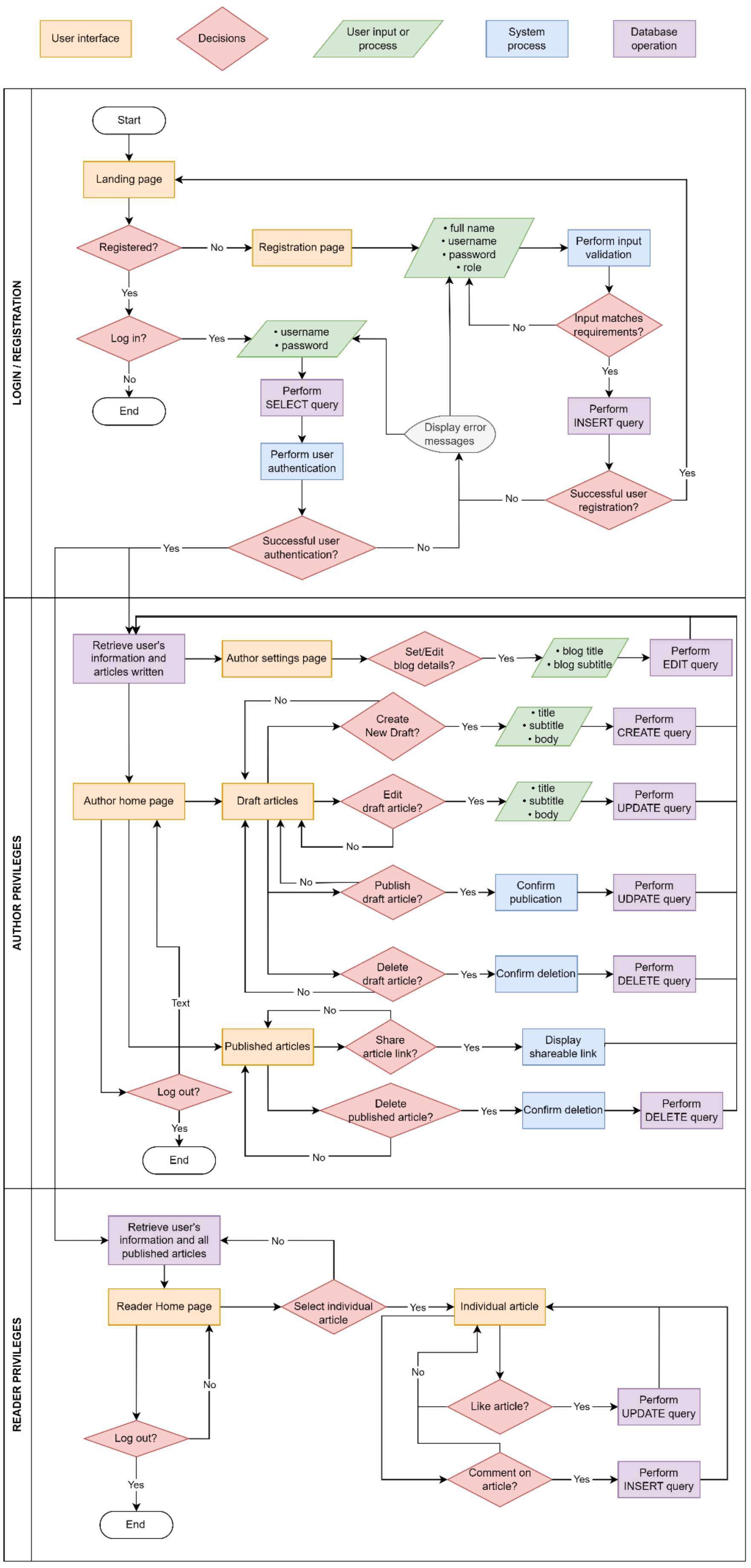
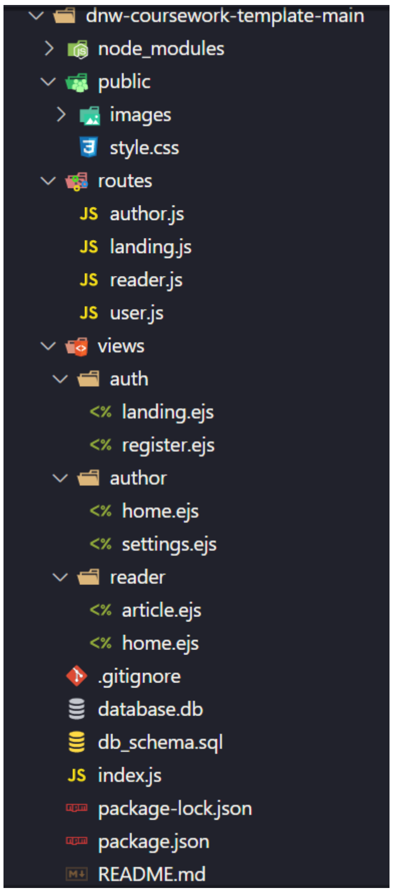


Figure 2. Page flow diagram for ScribbleBlog

Folder structure of project



```
⌄ dnw-coursework-template-main
  › node_modules
  ⌄ public
    › images
      style.css
  ⌄ routes
      author.js
      landing.js
      reader.js
      user.js
  ⌄ views
    ⌄ auth
        landing.ejs
        register.ejs
    ⌄ author
        home.ejs
        settings.ejs
    ⌄ reader
        article.ejs
        home.ejs
    .gitignore
    database.db
    db_schema.sql
    index.js
    package-lock.json
    package.json
    README.md
```

Figure 3. Folder structure of the project

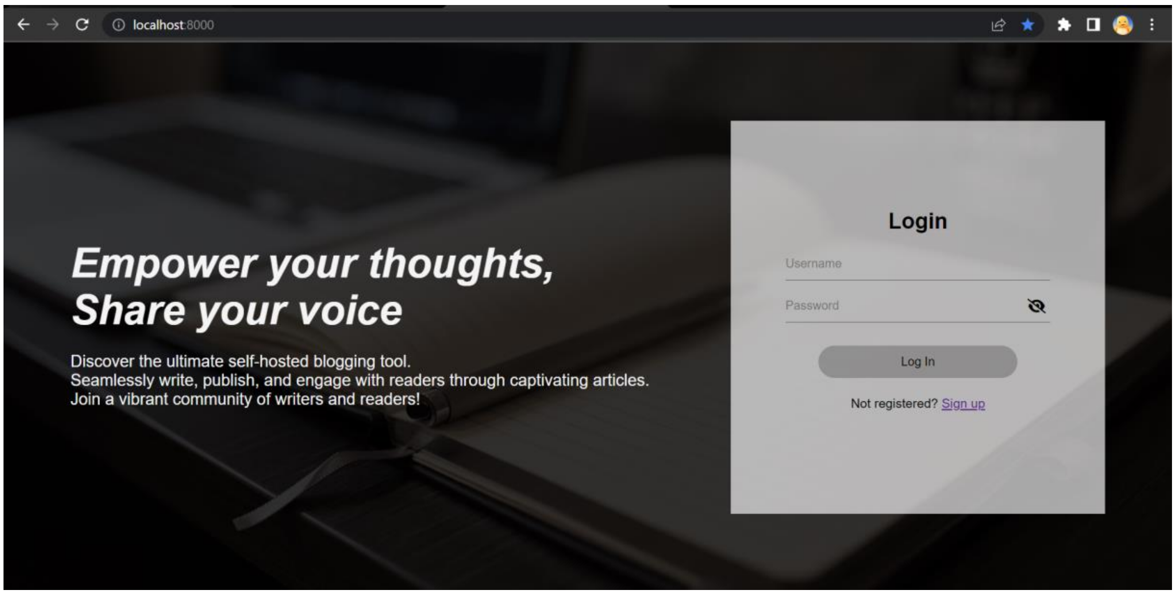
Folders and files that were added on top of the base template are elaborated further, as shown in the table below.

Folder		File	Purpose
public	**images**	main_bg.jpg	Image used for background of the landing page
		style.css	The styling of all pages is stored here
routes		landing.js	Routes for landing page is defined and handled here.
		user.js	Routes for *POST* method of login and registration are defined and handled here.
		author.js	Routes for all author related functionalities are defined and handled here.
		reader.js	Routes for all reader related functionalities are defined and handled here.
views	**auth**	landing.ejs*	Landing page. Displays login form.
		register.ejs*	Displays registration form.
	author	home.ejs*	Author home page. Displays list of draft and published articles. Includes options to create new, edit, publish, and delete draft articles, as well as share and delete published articles.
		settings.ejs*	Includes options to set or edit the blog title and subtitle.
	reader	home.ejs*	Reader home page. Displays list of published articles.
		article.ejs*	Displays article details depending on the article clicked. Includes options to like, dislike, and comment on the article.

**Toggle the links to see how the specific ejs file looks like.*

Screenshots

Landing page

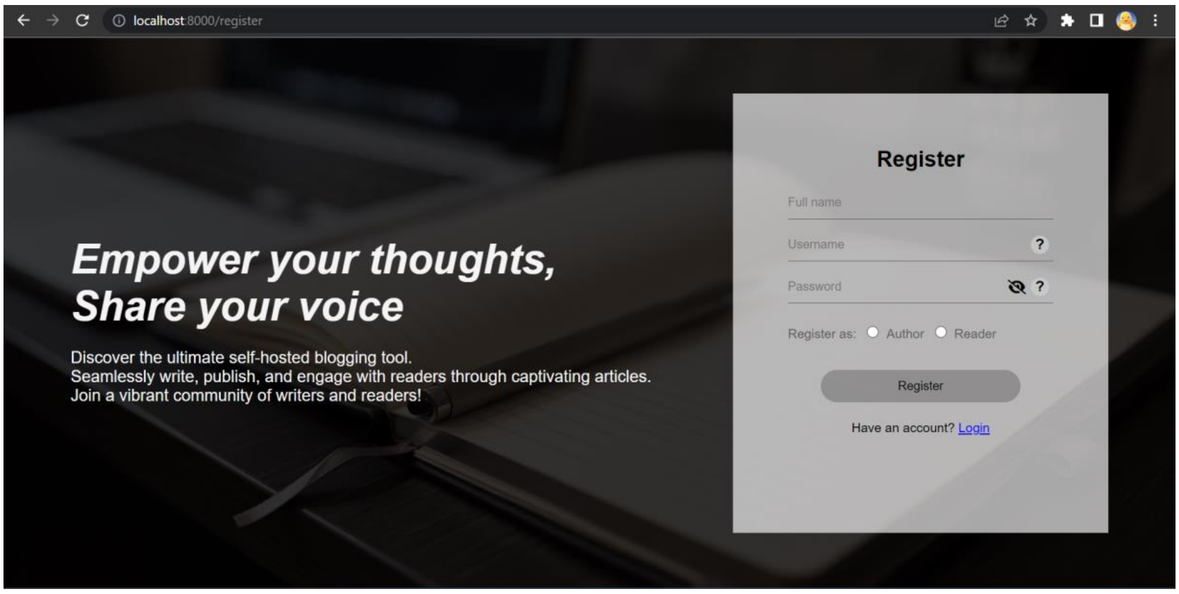


localhost:8000

Empower your thoughts, Share your voice

Discover the ultimate self-hosted blogging tool.
Seamlessly write, publish, and engage with readers through captivating articles.
Join a vibrant community of writers and readers!

Registration page

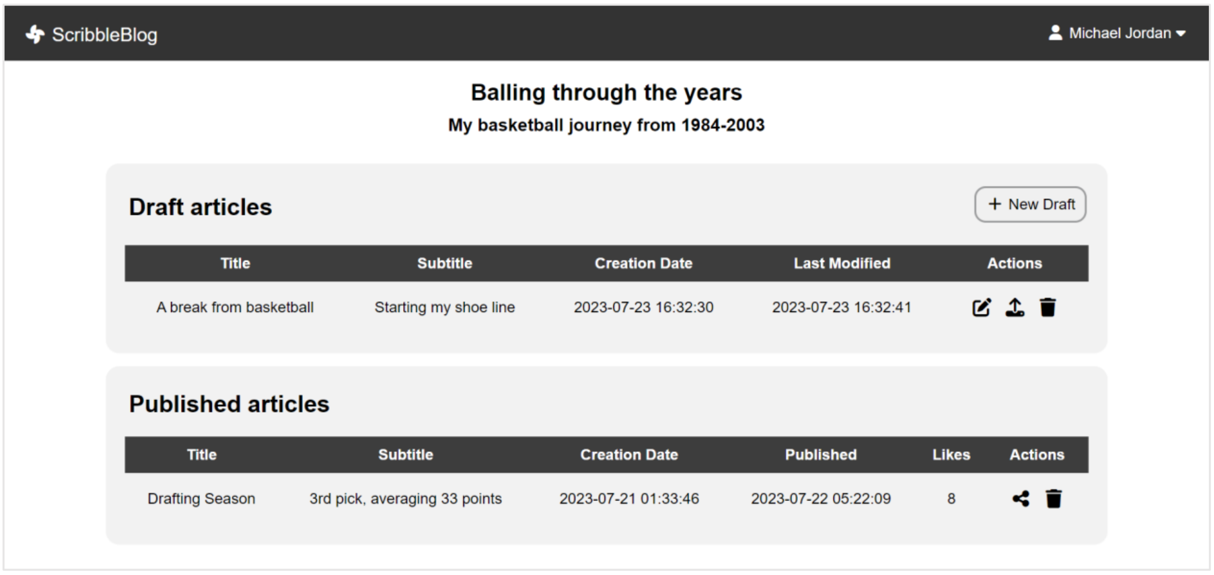


localhost:8000/register

Empower your thoughts, Share your voice

Discover the ultimate self-hosted blogging tool.
Seamlessly write, publish, and engage with readers through captivating articles.
Join a vibrant community of writers and readers!

Author's home page



Balling through the years

My basketball journey from 1984-2003

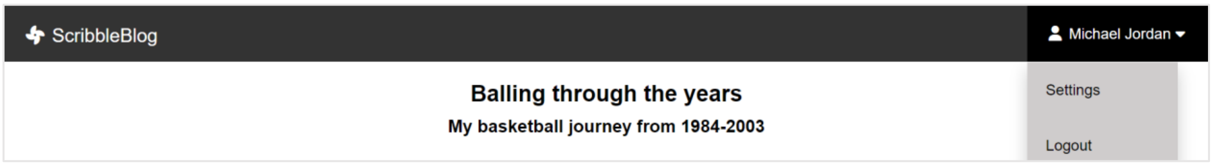
Draft articles

Title	Subtitle	Creation Date	Last Modified	Actions
A break from basketball	Starting my shoe line	2023-07-23 16:32:30	2023-07-23 16:32:41	

Published articles

Title	Subtitle	Creation Date	Published	Likes	Actions
Drafting Season	3rd pick, averaging 33 points	2023-07-21 01:33:46	2023-07-22 05:22:09	8	

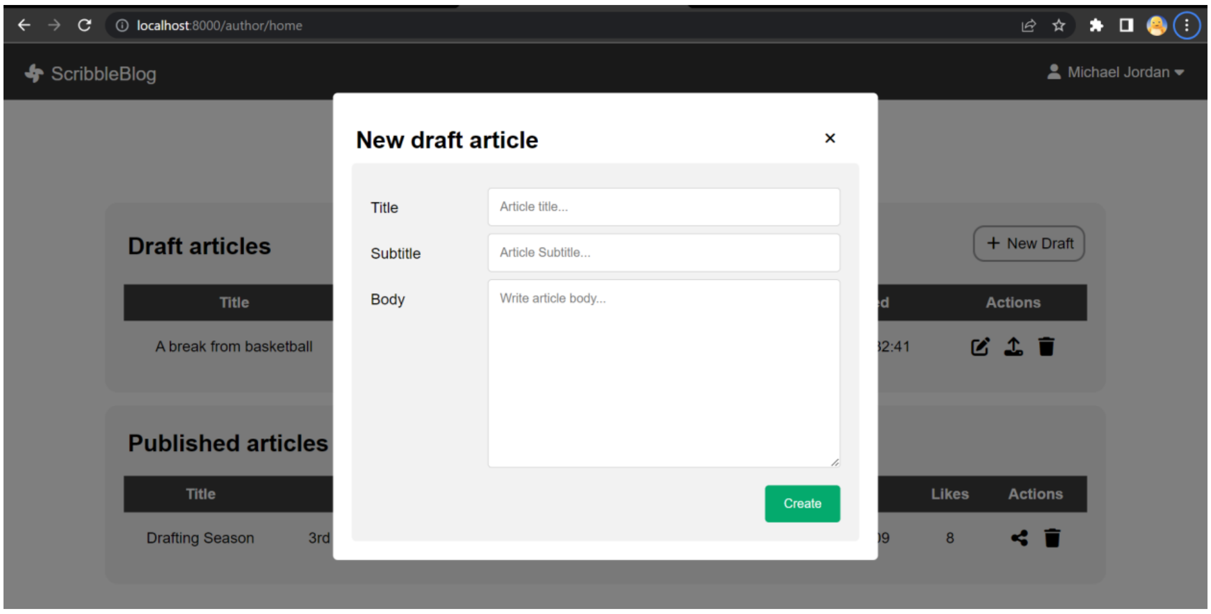
Navigation bar



Balling through the years

My basketball journey from 1984-2003

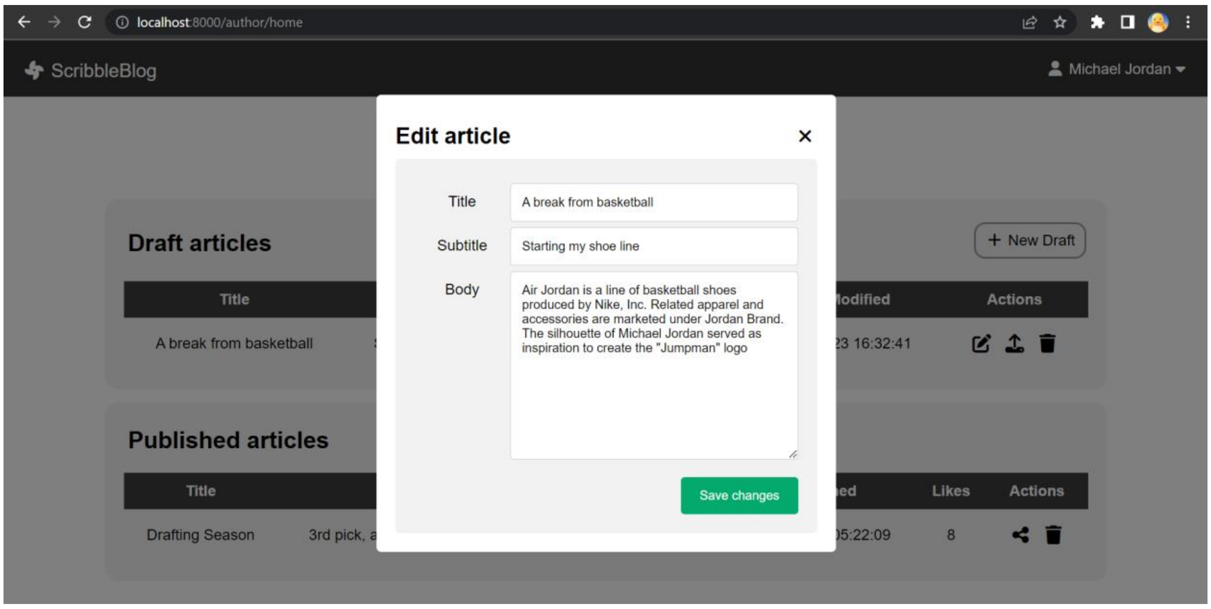
Create new draft modal



Title	Actions
A break from basketball	

Title	Likes	Actions
Drafting Season	8	

Edit draft article modal



Title: A break from basketball

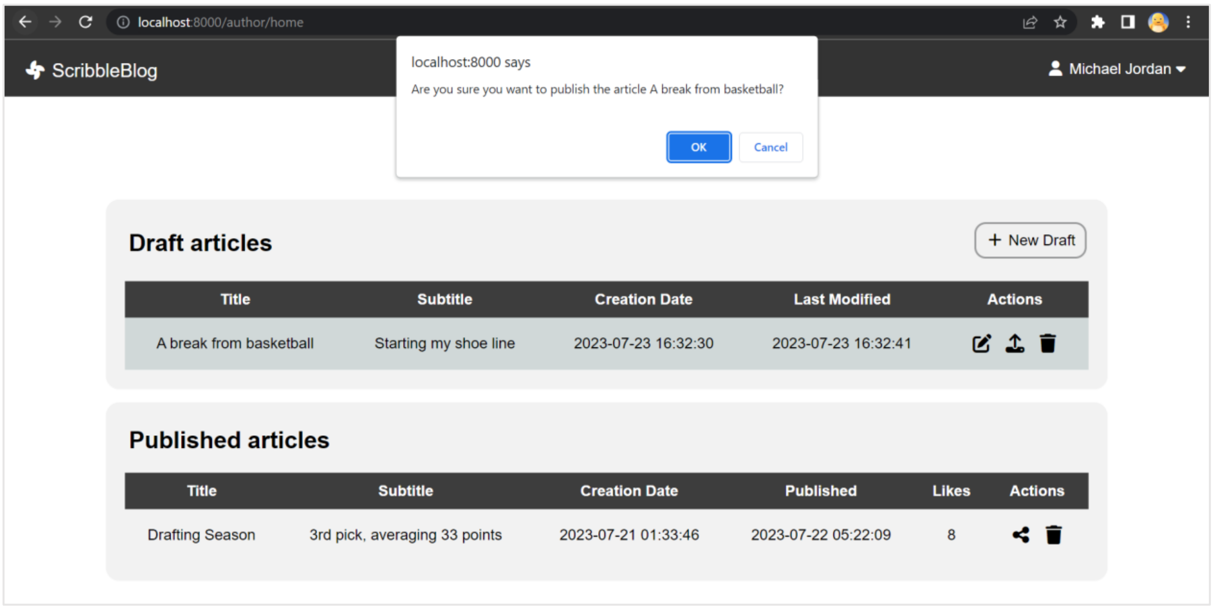
Subtitle: Starting my shoe line

Body: Air Jordan is a line of basketball shoes produced by Nike, Inc. Related apparel and accessories are marketed under Jordan Brand. The silhouette of Michael Jordan served as inspiration to create the "Jumpman" logo

Title	Modified	Actions
A break from basketball	3 16:32:41	

Title		Likes	Actions
Drafting Season	3rd pick, a	8	

Publish article confirmation window



localhost:8000/author/home

ScribbleBlog

Michael Jordan

localhost:8000 says

Are you sure you want to publish the article A break from basketball?

OK Cancel

Draft articles

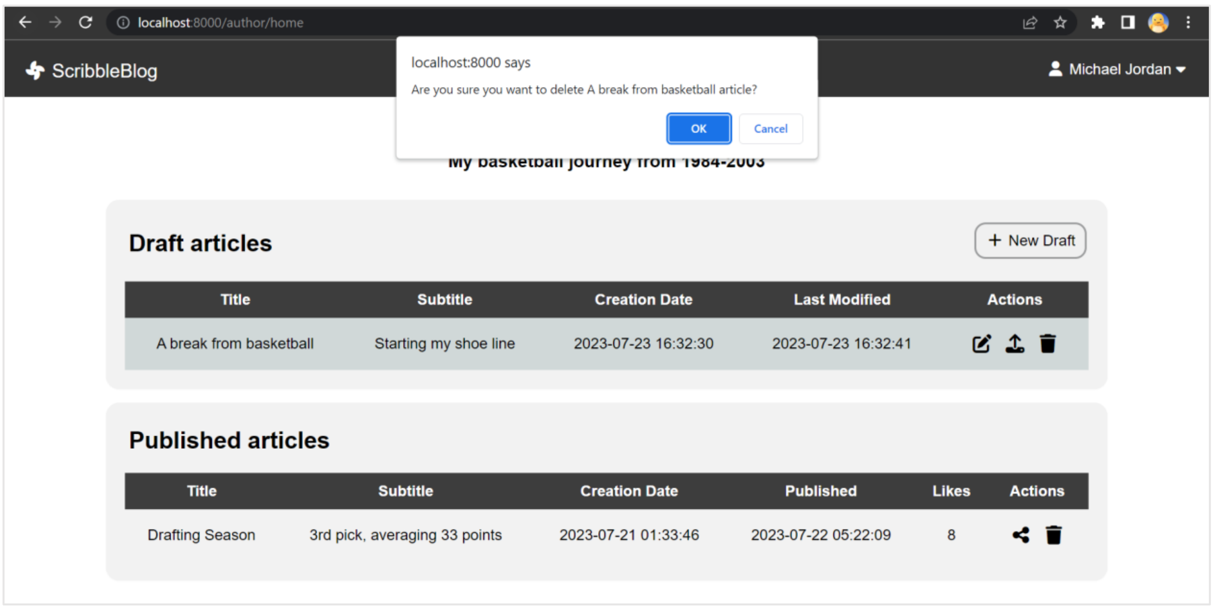
+ New Draft

Title	Subtitle	Creation Date	Last Modified	Actions
A break from basketball	Starting my shoe line	2023-07-23 16:32:30	2023-07-23 16:32:41	

Published articles

Title	Subtitle	Creation Date	Published	Likes	Actions
Drafting Season	3rd pick, averaging 33 points	2023-07-21 01:33:46	2023-07-22 05:22:09	8	

Delete draft article confirmation window



localhost:8000/author/home

ScribbleBlog

Michael Jordan

localhost:8000 says

Are you sure you want to delete A break from basketball article?

OK Cancel

My basketball journey from 1984-2003

Draft articles

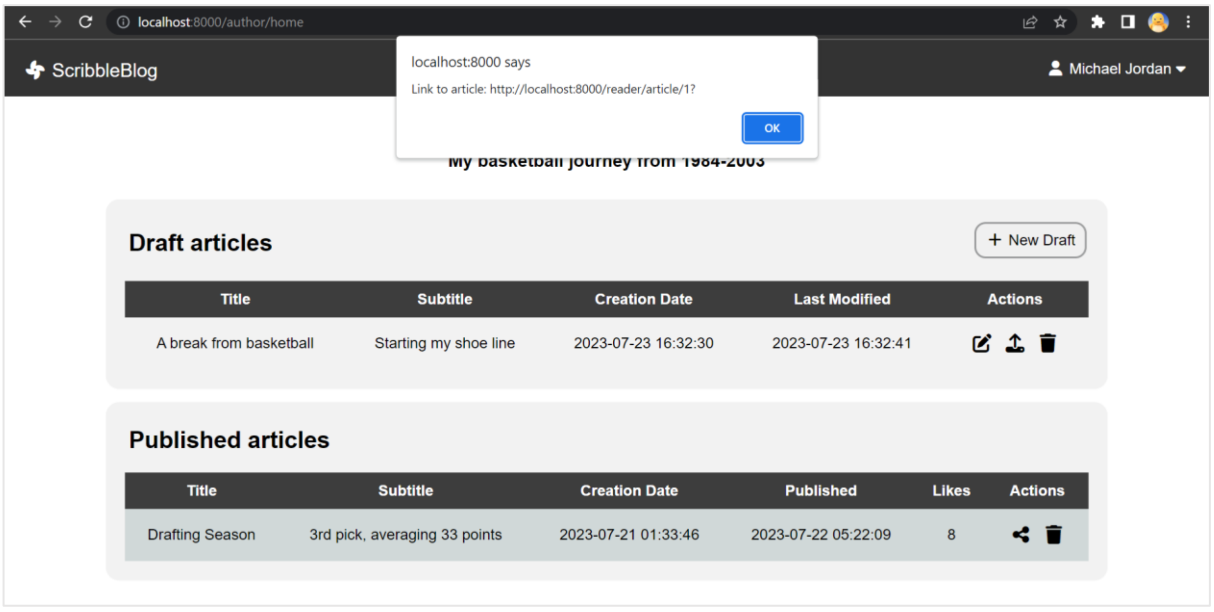
+ New Draft

Title	Subtitle	Creation Date	Last Modified	Actions
A break from basketball	Starting my shoe line	2023-07-23 16:32:30	2023-07-23 16:32:41	

Published articles

Title	Subtitle	Creation Date	Published	Likes	Actions
Drafting Season	3rd pick, averaging 33 points	2023-07-21 01:33:46	2023-07-22 05:22:09	8	

Share link alert window



localhost:8000/author/home

ScribbleBlog

Michael Jordan

localhost:8000 says

Link to article: http://localhost:8000/reader/article/1?

OK

My basketball journey from 1984-2003

Draft articles

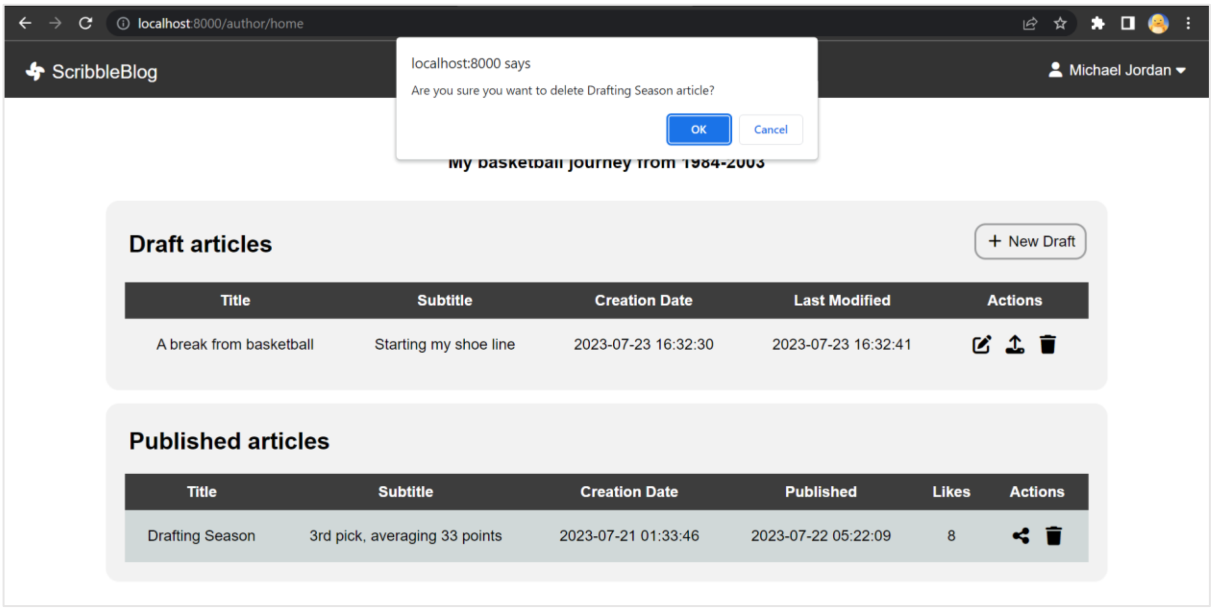
+ New Draft

Title	Subtitle	Creation Date	Last Modified	Actions
A break from basketball	Starting my shoe line	2023-07-23 16:32:30	2023-07-23 16:32:41	

Published articles

Title	Subtitle	Creation Date	Published	Likes	Actions
Drafting Season	3rd pick, averaging 33 points	2023-07-21 01:33:46	2023-07-22 05:22:09	8	

Delete published article confirmation window



localhost:8000/author/home

ScribbleBlog

Michael Jordan

localhost:8000 says

Are you sure you want to delete Drafting Season article?

OK Cancel

My basketball journey from 1984-2003

Draft articles

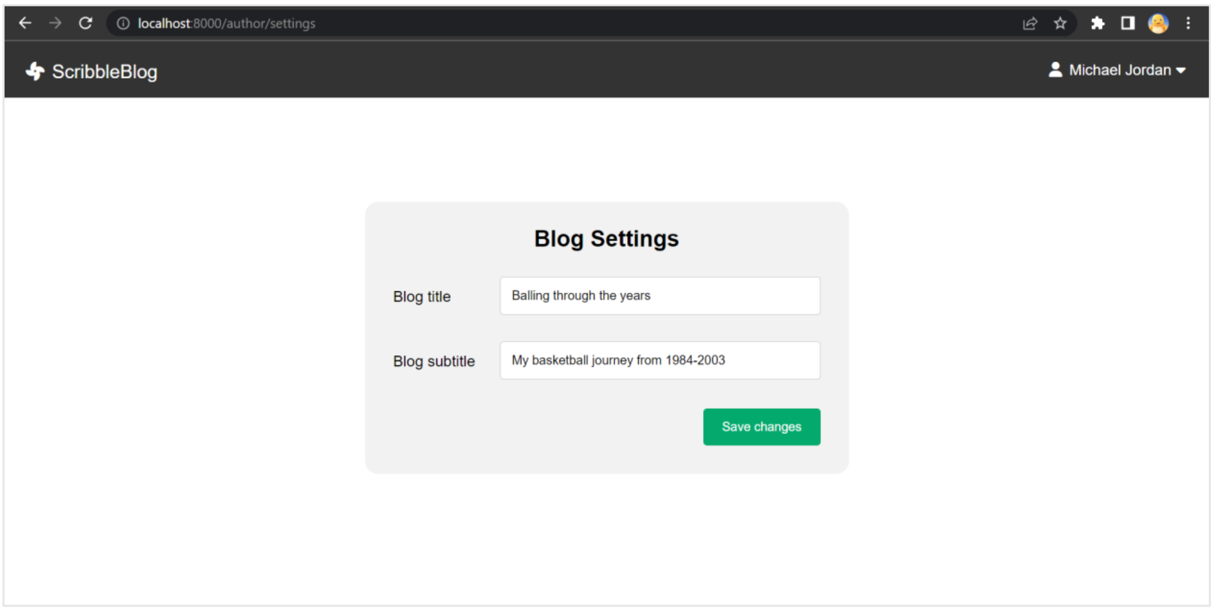
+ New Draft

Title	Subtitle	Creation Date	Last Modified	Actions
A break from basketball	Starting my shoe line	2023-07-23 16:32:30	2023-07-23 16:32:41	

Published articles

Title	Subtitle	Creation Date	Published	Likes	Actions
Drafting Season	3rd pick, averaging 33 points	2023-07-21 01:33:46	2023-07-22 05:22:09	8	

Author’s settings page



localhost:8000/author/settings

ScribbleBlog

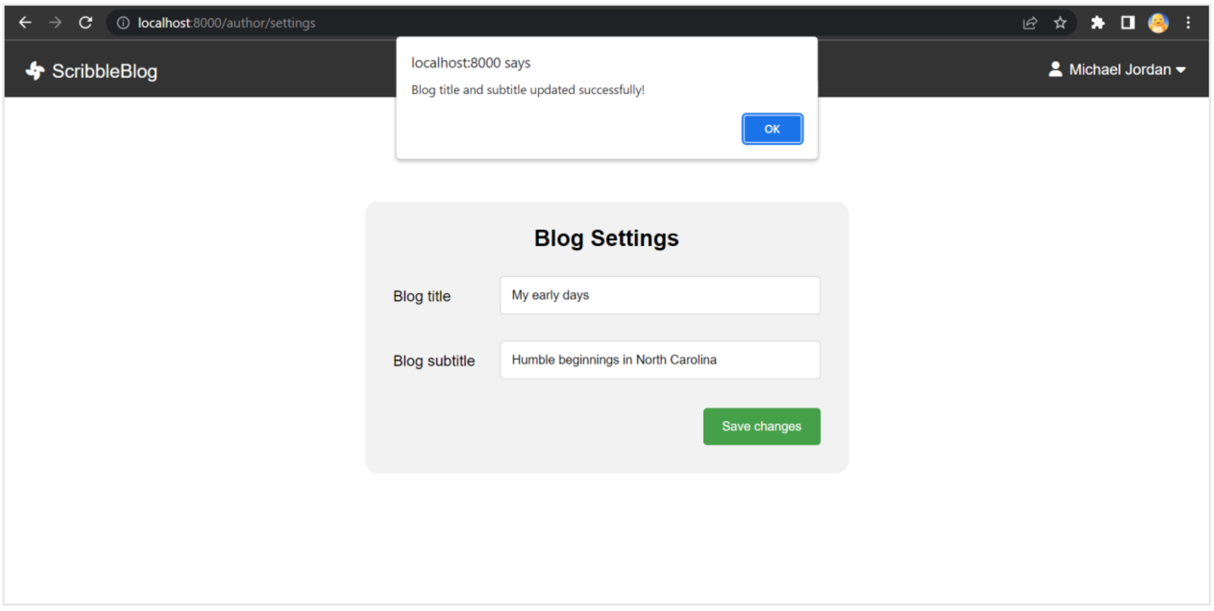
Michael Jordan

Blog Settings

Blog title	Balling through the years
Blog subtitle	My basketball journey from 1984-2003

Save changes

Successful update alert window



localhost:8000/author/settings

ScribbleBlog

Michael Jordan

localhost:8000 says

Blog title and subtitle updated successfully!

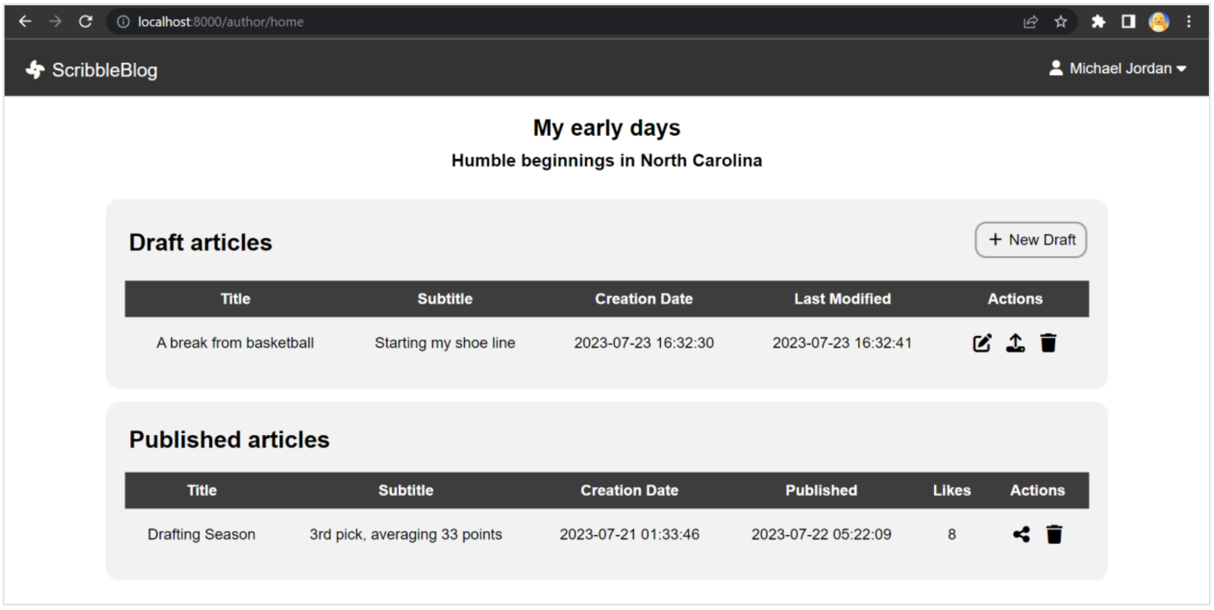
OK

Blog Settings

Blog title	My early days
Blog subtitle	Humble beginnings in North Carolina

Save changes

Successful update of blog title and subtitle



localhost:8000/author/home

ScribbleBlog

Michael Jordan

My early days

Humble beginnings in North Carolina

Draft articles

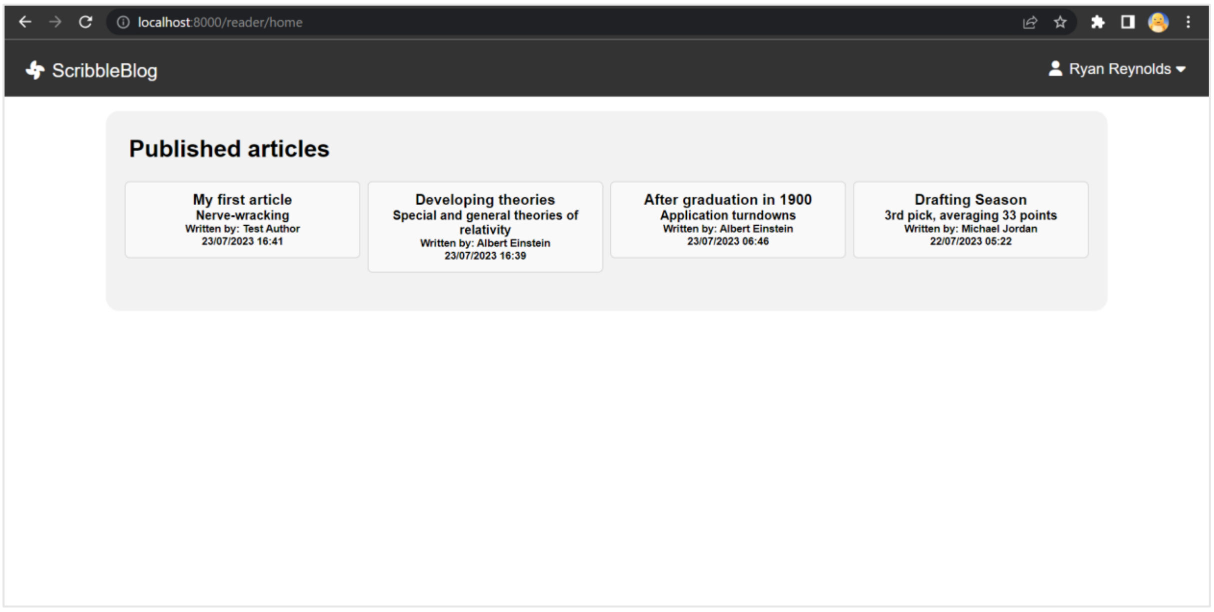
+ New Draft

Title	Subtitle	Creation Date	Last Modified	Actions
A break from basketball	Starting my shoe line	2023-07-23 16:32:30	2023-07-23 16:32:41	

Published articles

Title	Subtitle	Creation Date	Published	Likes	Actions
Drafting Season	3rd pick, averaging 33 points	2023-07-21 01:33:46	2023-07-22 05:22:09	8	

Reader's home page



localhost:8000/reader/home

ScribbleBlog

Ryan Reynolds

Published articles

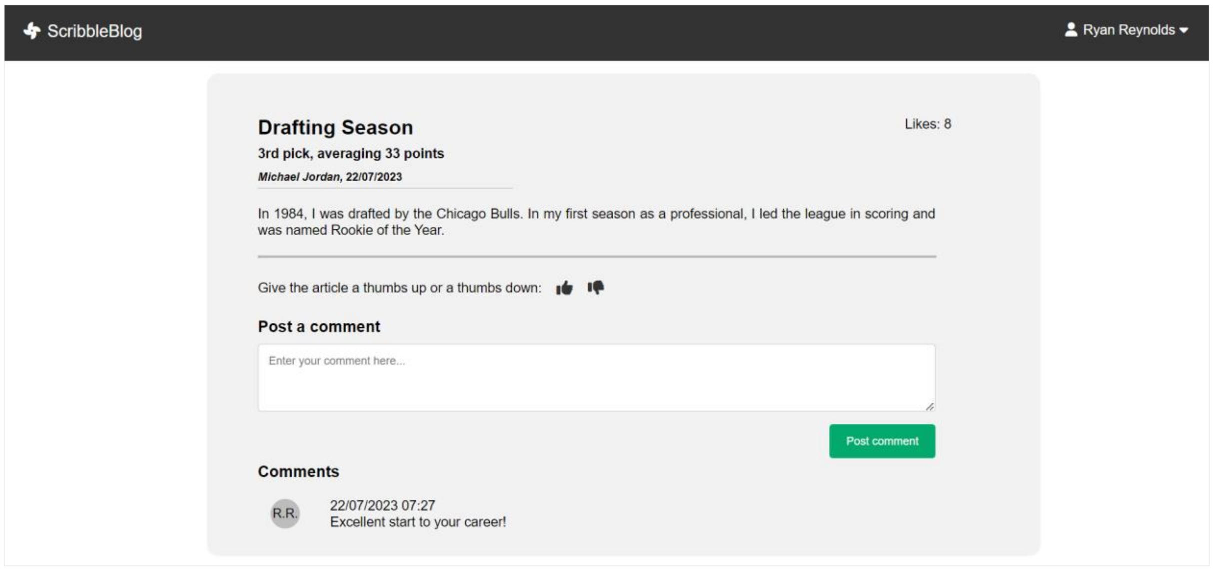
My first article
Nerve-wracking
Written by: Test Author
23/07/2023 16:41

Developing theories
Special and general theories of relativity
Written by: Albert Einstein
23/07/2023 16:39

After graduation in 1900
Application turndowns
Written by: Albert Einstein
23/07/2023 06:46

Drafting Season
3rd pick, averaging 33 points
Written by: Michael Jordan
22/07/2023 05:22

Individual article's page



ScribbleBlog

Ryan Reynolds

Drafting Season

Likes: 8

3rd pick, averaging 33 points

Michael Jordan, 22/07/2023

In 1984, I was drafted by the Chicago Bulls. In my first season as a professional, I led the league in scoring and was named Rookie of the Year.

Give the article a thumbs up or a thumbs down:

Post a comment

Enter your comment here...

Post comment

Comments

R.R. 22/07/2023 07:27
Excellent start to your career!

Extension write-up

Chosen extension

Password access for author pages and routes

Flow diagram of Login/Registration process

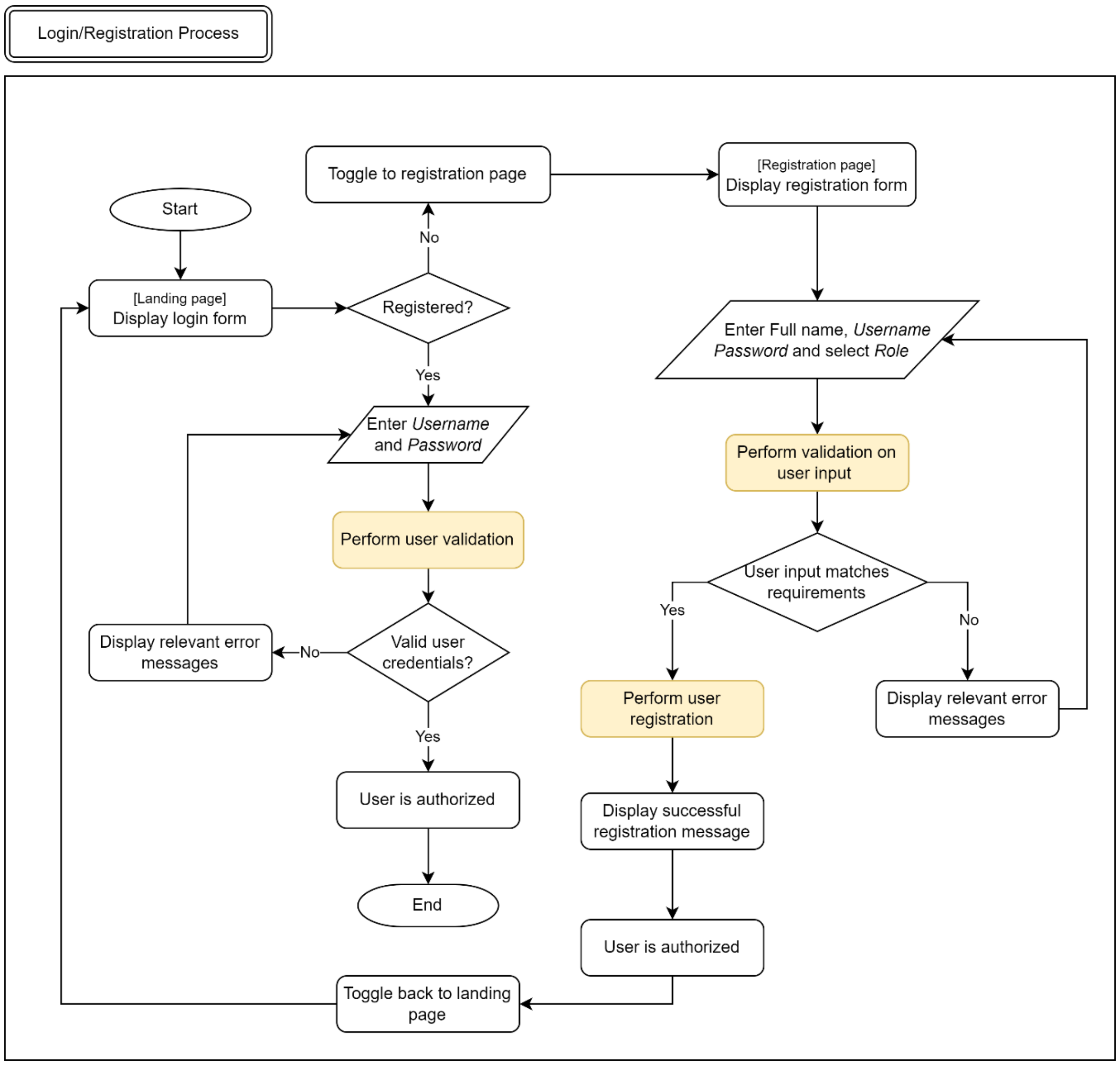

Login/Registration Process
Start
[Landing page] Display login form
Registered?
No
Yes
Toggle to registration page
[Registration page] Display registration form
Enter Full name, Username Password and select Role
Perform validation on user input
User input matches requirements
Yes
No
Perform user registration
Display relevant error messages
Display successful registration message
User is authorized
Enter Username and Password
Perform user validation
Valid user credentials?
No
Yes
Display relevant error messages
User is authorized
End
Toggle back to landing page

Figure 2. Flow diagram of login and registration process

Figure 2 depicts the flow diagram of the login and registration process.

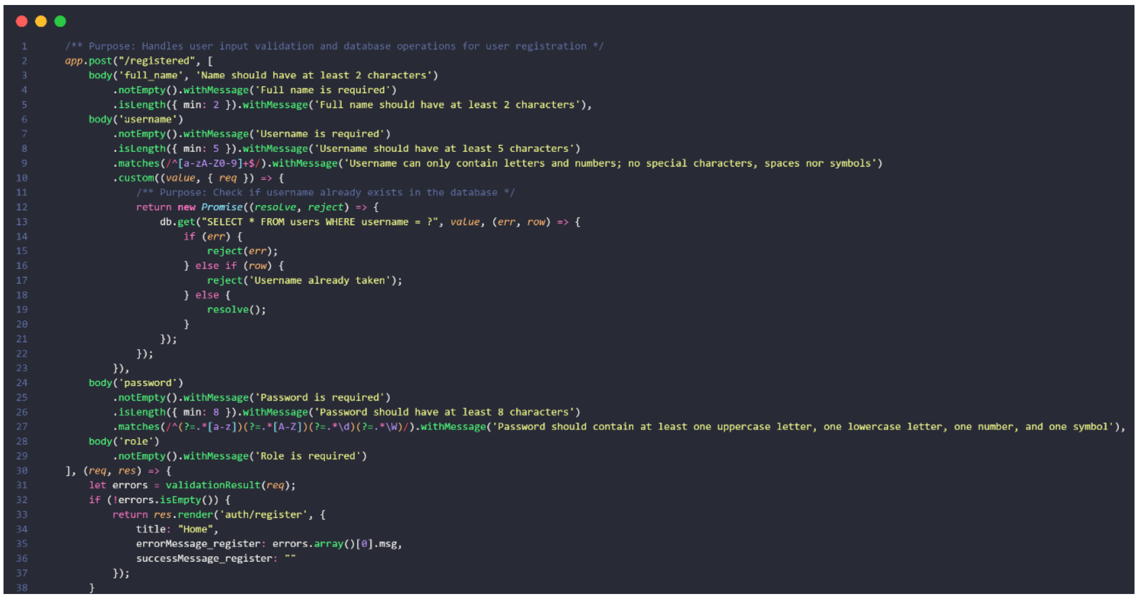
The process starts at the landing page where the login form is displayed. The user can toggle between the between login and registration forms by clicking on the *Sign up* and *Login* links.

Registration Process

1. Toggle to registration page
    - The user first toggles to the registration page to create an account.
2. Complete the registration form
    - Full name, username, password, and role are provided by the user.
3. Submit form
    - Upon submission—clicking on Register button—, the frontend would send a *POST* request to the middleware with all the user's input from step 2.

4. Middleware handles request
    - Once the request is received, the user's input is validated and checked if the requirement for each field is met. This can be seen in the code snippet below.

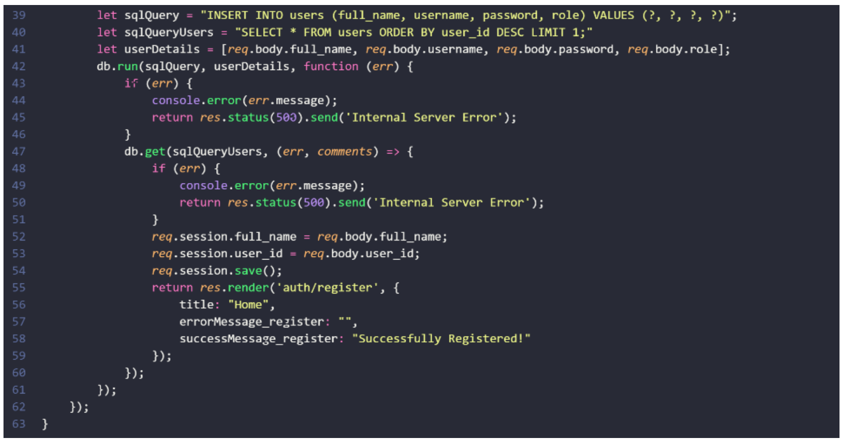
Code line	Input validation for
3 to 5	Full name
6 to 23	Username [In lines 10 to 23, a *SELECT* database operation is done to check if the username already exists in the database]
24 to 27	Password
28 to 29	Role



```
1       /** Purpose: Handles user input validation and database operations for user registration */
2       app.post("/registered", [
3           body('full_name', 'Name should have at least 2 characters')
4               .notEmpty().withMessage('Full name is required')
5               .isLength({ min: 2 }).withMessage('Full name should have at least 2 characters'),
6           body('username')
7               .notEmpty().withMessage('Username is required')
8               .isLength({ min: 5 }).withMessage('Username should have at least 5 characters')
9               .matches(/^[a-zA-Z0-9]+$/).withMessage('Username can only contain letters and numbers; no special characters, spaces nor symbols')
10              .custom((value, { req }) => {
11                  /** Purpose: Check if username already exists in the database */
12                  return new Promise((resolve, reject) => {
13                      db.get("SELECT * FROM users WHERE username = ?", value, (err, row) => {
14                          if (err) {
15                              reject(err);
16                          } else if (row) {
17                              reject('Username already taken');
18                          } else {
19                              resolve();
20                          }
21                      });
22                  });
23              }),
24          body('password')
25              .notEmpty().withMessage('Password is required')
26              .isLength({ min: 8 }).withMessage('Password should have at least 8 characters')
27              .matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*\W)/).withMessage('Password should contain at least one uppercase letter, one lowercase letter, one number, and one symbol'),
28          body('role')
29              .notEmpty().withMessage('Role is required')
30      ], (req, res) => {
31          let errors = validationResult(req);
32          if (!errors.isEmpty()) {
33              return res.render('auth/register', {
34                  title: "Home",
35                  errorMessage_register: errors.array()[0].msg,
36                  successMessage_register: ""
37              });
38          }
```

- Any errors received, would result in rendering the registration page with the corresponding error messages displayed to the user.

5. Database operation is performed
    - If the user inputs are valid, a new user record would be created in the database using an *INSERT* query, this can be seen in line 39 below.



```
39          let sqlQuery = "INSERT INTO users (full_name, username, password, role) VALUES (?, ?, ?, ?)";
40          let sqlQueryUsers = "SELECT * FROM users ORDER BY user_id DESC LIMIT 1;"
41          let userDetails = [req.body.full_name, req.body.username, req.body.password, req.body.role];
42          db.run(sqlQuery, userDetails, function (err) {
43              if (err) {
44                  console.error(err.message);
45                  return res.status(500).send('Internal Server Error');
46              }
47              db.get(sqlQueryUsers, (err, comments) => {
48                  if (err) {
49                      console.error(err.message);
50                      return res.status(500).send('Internal Server Error');
51                  }
52                  req.session.full_name = req.body.full_name;
53                  req.session.user_id = req.body.user_id;
54                  req.session.save();
55                  return res.render('auth/register', {
56                      title: "Home",
57                      errorMessage_register: "",
58                      successMessage_register: "Successfully Registered!"
59                  });
60              });
61          });
62      });
63  }
```

6. User is now authorized
    - With the successful insertion of user data into the database, the user is now authorized to access pages within their role.

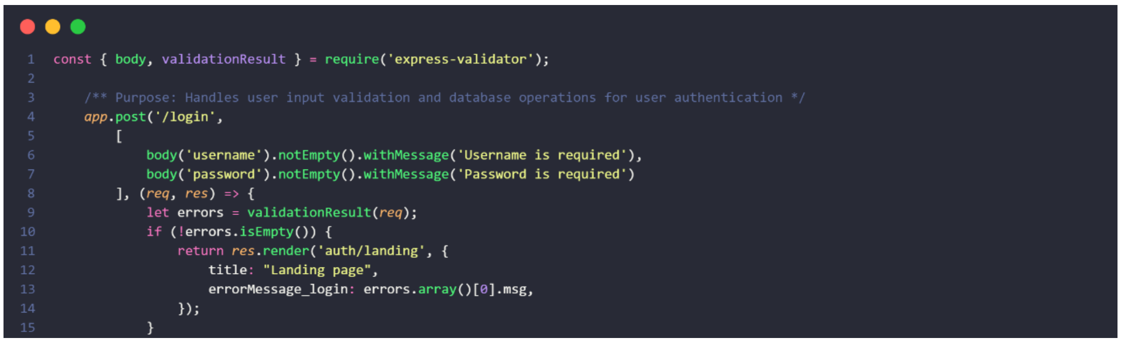
Login Process

1. Toggle to login page
    - The user first accesses the landing page to login, or toggles *login* from registration page.
2. Complete the registration form
    - Username and password are provided by the user.
3. Submit form

- Upon submission—clicking on Log In button—, the frontend would send a *POST* request to the middleware with all the user's login credentials from step 2.

4. Middleware handles request
   - Once the request is received, the user's input is validated and checked if it is empty or not. This can be seen in the code snippet below.

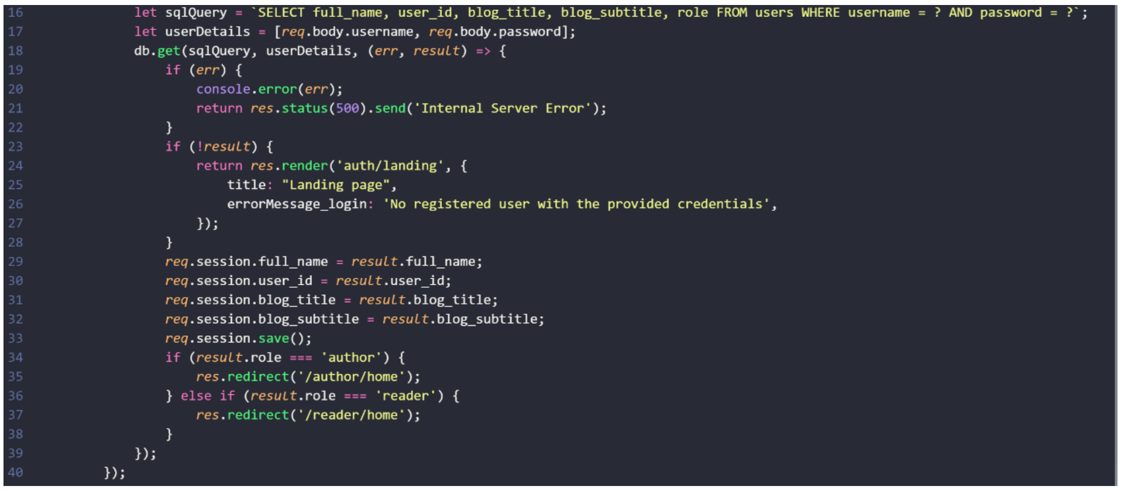
Code line	Input validation for
6	Username
7	Password



```
1   const { body, validationResult } = require('express-validator');
2
3       /** Purpose: Handles user input validation and database operations for user authentication */
4       app.post('/login',
5           [
6               body('username').notEmpty().withMessage('Username is required'),
7               body('password').notEmpty().withMessage('Password is required')
8           ], (req, res) => {
9               let errors = validationResult(req);
10              if (!errors.isEmpty()) {
11                  return res.render('auth/landing', {
12                      title: "Landing page",
13                      errorMessage_login: errors.array()[0].msg,
14                  });
15              }
```

   - Any errors received, would result in rendering the landing page with the corresponding error messages displayed to the user.

5. Database operation is performed
   - If the user inputs are valid, the user record would be retrieved from the database using a *SELECT* query, this can be seen in line 16 below.



```
16              let sqlQuery = `SELECT full_name, user_id, blog_title, blog_subtitle, role FROM users WHERE username = ? AND password = ?`;
17              let userDetails = [req.body.username, req.body.password];
18              db.get(sqlQuery, userDetails, (err, result) => {
19                  if (err) {
20                      console.error(err);
21                      return res.status(500).send('Internal Server Error');
22                  }
23                  if (!result) {
24                      return res.render('auth/landing', {
25                          title: "Landing page",
26                          errorMessage_login: 'No registered user with the provided credentials',
27                      });
28                  }
29                  req.session.full_name = result.full_name;
30                  req.session.user_id = result.user_id;
31                  req.session.blog_title = result.blog_title;
32                  req.session.blog_subtitle = result.blog_subtitle;
33                  req.session.save();
34                  if (result.role === 'author') {
35                      res.redirect('/author/home');
36                  } else if (result.role === 'reader') {
37                      res.redirect('/reader/home');
38                  }
39              });
40          });
```

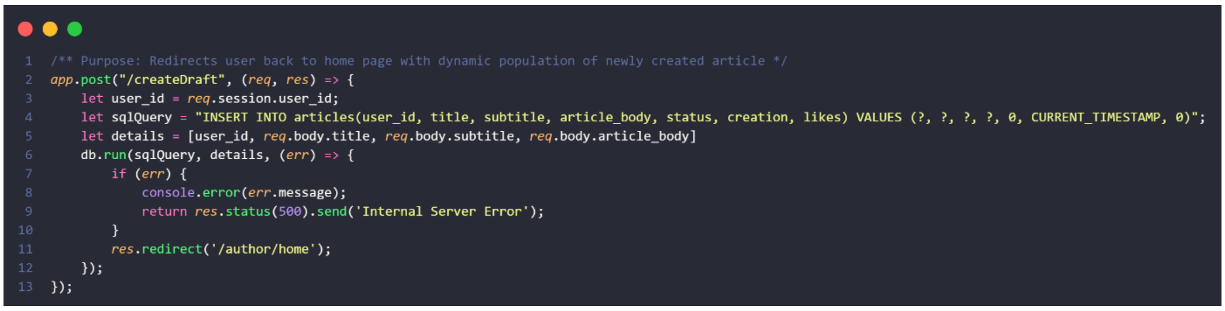
   - Lines 29 to 33 saves the user's login credentials so that the middleware can track their session data for future HTTP requests.

6. Access granted
   - With the successful authentication, the user is now logged in.

Middleware code snippets

Author functionalities

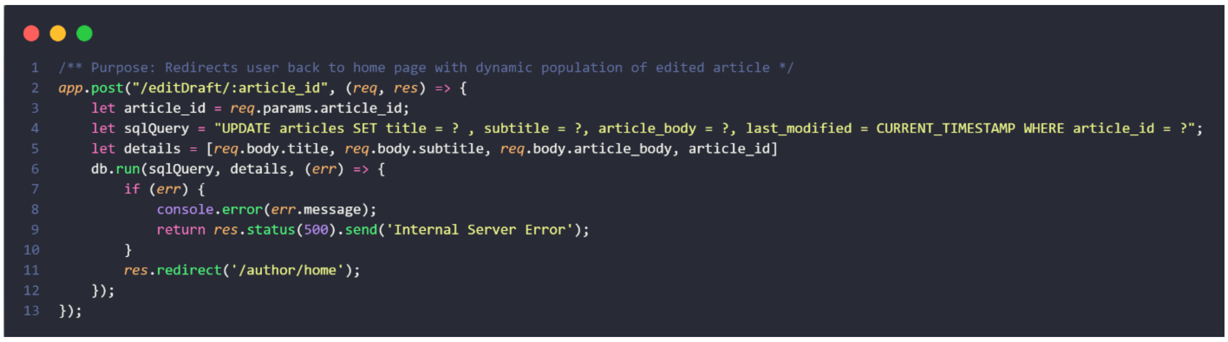
Creating new draft article



```
1  /** Purpose: Redirects user back to home page with dynamic population of newly created article */
2  app.post("/createDraft", (req, res) => {
3      let user_id = req.session.user_id;
4      let sqlQuery = "INSERT INTO articles(user_id, title, subtitle, article_body, status, creation, likes) VALUES (?, ?, ?, ?, 0, CURRENT_TIMESTAMP, 0)";
5      let details = [user_id, req.body.title, req.body.subtitle, req.body.article_body]
6      db.run(sqlQuery, details, (err) => {
7          if (err) {
8              console.error(err.message);
9              return res.status(500).send('Internal Server Error');
10         }
11         res.redirect('/author/home');
12     });
13 });
```

With the stored user ID in the session—as seen in line 3—an INSERT query is performed to respond to the author’s request of creating a new draft article. The form to create a new draft article has input fields that are passed into the query, as an input parameter to be stored in the database.

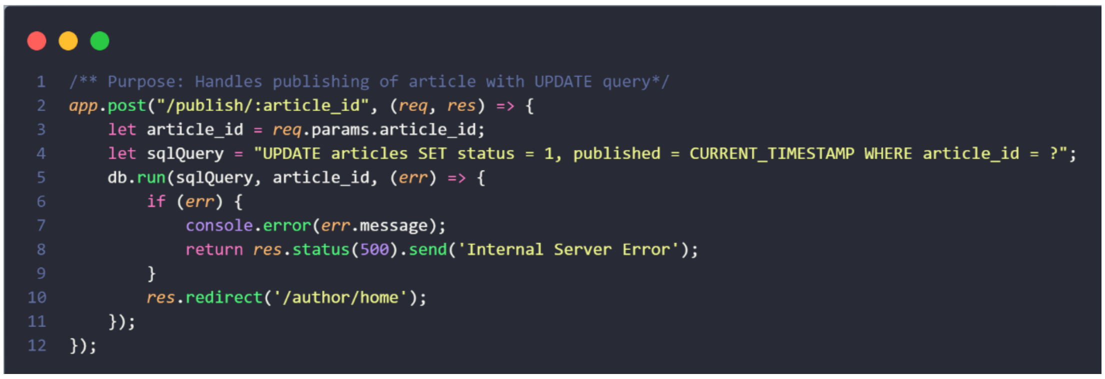
Edit draft article



```
1  /** Purpose: Redirects user back to home page with dynamic population of edited article */
2  app.post("/editDraft/:article_id", (req, res) => {
3      let article_id = req.params.article_id;
4      let sqlQuery = "UPDATE articles SET title = ? , subtitle = ?, article_body = ?, last_modified = CURRENT_TIMESTAMP WHERE article_id = ?";
5      let details = [req.body.title, req.body.subtitle, req.body.article_body, article_id]
6      db.run(sqlQuery, details, (err) => {
7          if (err) {
8              console.error(err.message);
9              return res.status(500).send('Internal Server Error');
10         }
11         res.redirect('/author/home');
12     });
13 });
```

The article ID is extracted from the route parameters as seen in line 3. This is used as an input parameter for the database to know which article specifically to update. An *UPDATE* query is performed to set the revised fields based on the author’s input in the edit draft article form.

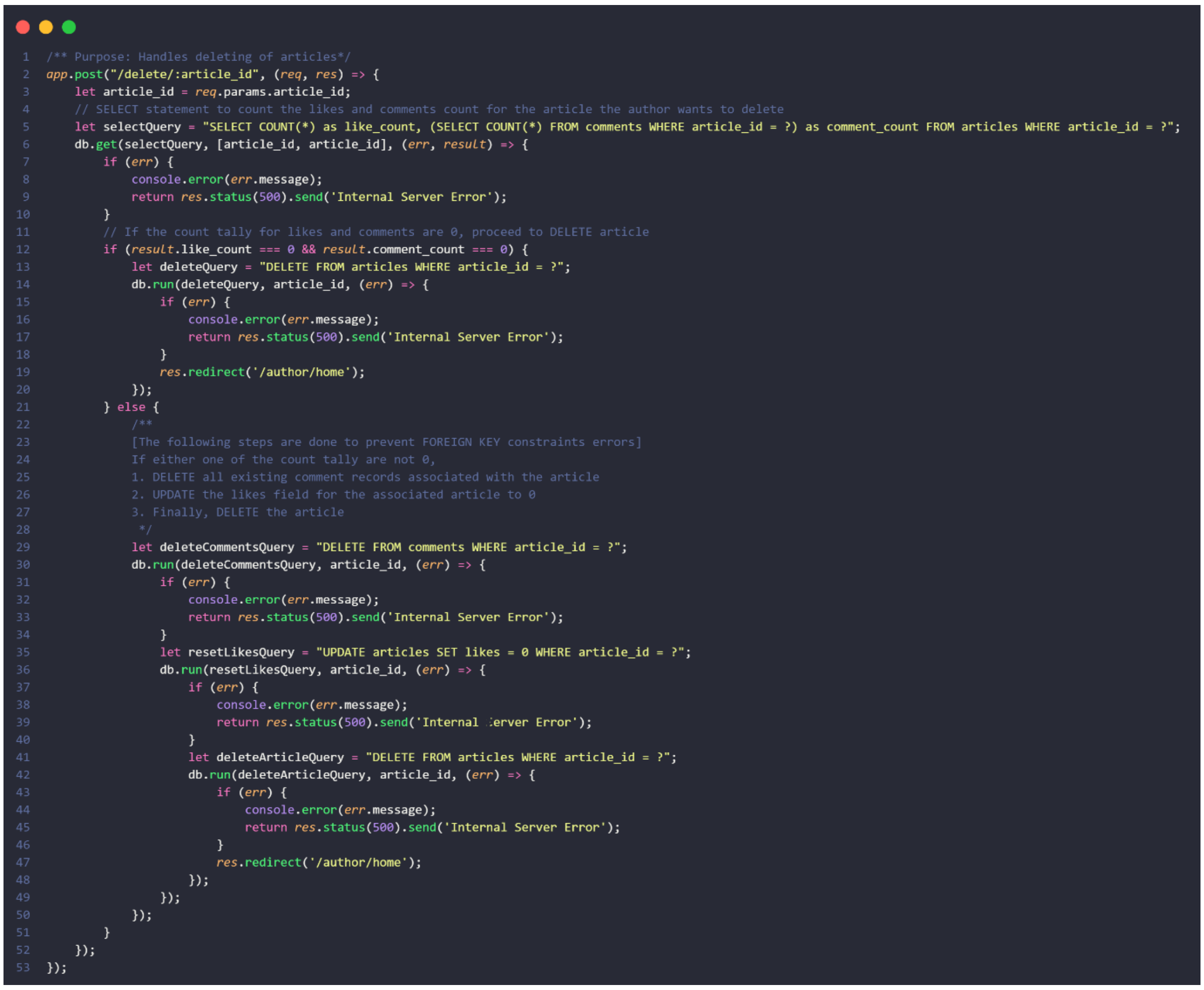
Publish draft article



```
1  /** Purpose: Handles publishing of article with UPDATE query*/
2  app.post("/publish/:article_id", (req, res) => {
3      let article_id = req.params.article_id;
4      let sqlQuery = "UPDATE articles SET status = 1, published = CURRENT_TIMESTAMP WHERE article_id = ?";
5      db.run(sqlQuery, article_id, (err) => {
6          if (err) {
7              console.error(err.message);
8              return res.status(500).send('Internal Server Error');
9          }
10         res.redirect('/author/home');
11     });
12 });
```

For context, when status is set to 0, the article is in *Draft* state, and when it is set to 1, the article is in *Published* state. So likewise, the article ID is extracted from the route parameters as seen in line 3. This is used as an input parameter for the database to know which article should have their status *SET* to *Published*.

Delete draft article



```
1   /** Purpose: Handles deleting of articles*/
2   app.post("/delete/:article_id", (req, res) => {
3       let article_id = req.params.article_id;
4       // SELECT statement to count the likes and comments count for the article the author wants to delete
5       let selectQuery = "SELECT COUNT(*) as like_count, (SELECT COUNT(*) FROM comments WHERE article_id = ?) as comment_count FROM articles WHERE article_id = ?";
6       db.get(selectQuery, [article_id, article_id], (err, result) => {
7           if (err) {
8               console.error(err.message);
9               return res.status(500).send('Internal Server Error');
10          }
11          // If the count tally for likes and comments are 0, proceed to DELETE article
12          if (result.like_count === 0 && result.comment_count === 0) {
13              let deleteQuery = "DELETE FROM articles WHERE article_id = ?";
14              db.run(deleteQuery, article_id, (err) => {
15                  if (err) {
16                      console.error(err.message);
17                      return res.status(500).send('Internal Server Error');
18                  }
19                  res.redirect('/author/home');
20              });
21          } else {
22              /**
23              [The following steps are done to prevent FOREIGN KEY constraints errors]
24              If either one of the count tally are not 0,
25              1. DELETE all existing comment records associated with the article
26              2. UPDATE the likes field for the associated article to 0
27              3. Finally, DELETE the article
28               */
29              let deleteCommentsQuery = "DELETE FROM comments WHERE article_id = ?";
30              db.run(deleteCommentsQuery, article_id, (err) => {
31                  if (err) {
32                      console.error(err.message);
33                      return res.status(500).send('Internal Server Error');
34                  }
35                  let resetLikesQuery = "UPDATE articles SET likes = 0 WHERE article_id = ?";
36                  db.run(resetLikesQuery, article_id, (err) => {
37                      if (err) {
38                          console.error(err.message);
39                          return res.status(500).send('Internal Server Error');
40                      }
41                      let deleteArticleQuery = "DELETE FROM articles WHERE article_id = ?";
42                      db.run(deleteArticleQuery, article_id, (err) => {
43                          if (err) {
44                              console.error(err.message);
45                              return res.status(500).send('Internal Server Error');
46                          }
47                          res.redirect('/author/home');
48                      });
49                  });
50              });
51          }
52      });
53  });
```

Error encountered when deleting

Initially, I received a *FOREIGN KEY constraint* error when deleting published articles. This occured when there are already likes and comments for that specific article. Otherwise, there would be no problems deleting it.

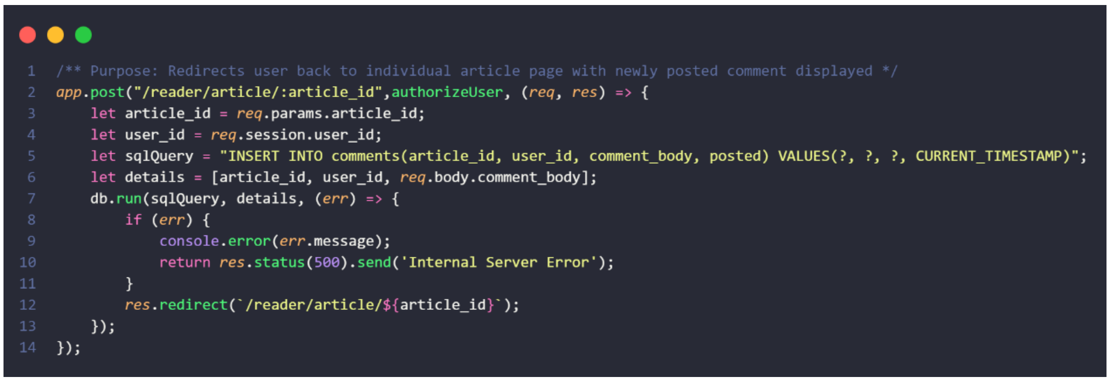
How the error was resolved

To resolve the error, I did the following steps:

1. First, do a *COUNT(*)* query to tally the likes and comments for the specific article (line 5)
2. If there are no likes or comments on that article, proceed with deletion (lines 12 to 20)
3. Else, delete corresponding comments first, then update likes to 0 (lines 29 to 47)
    - After which, proceed with the deletion

Reader functionalities

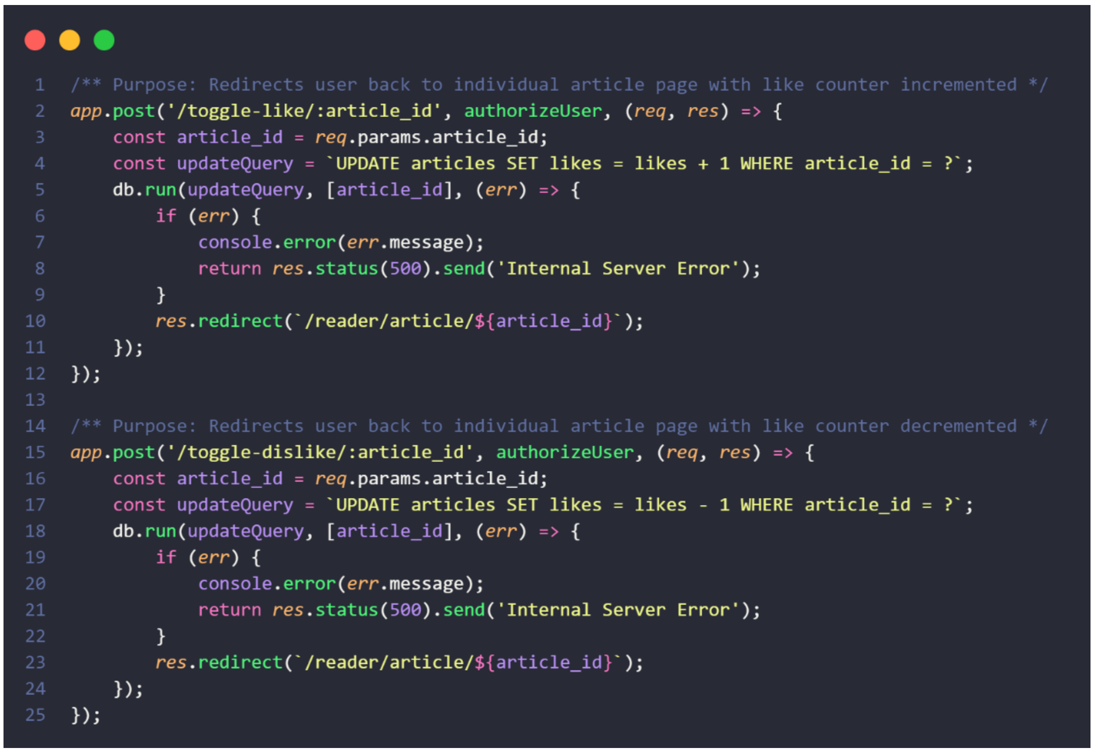
Comment function



```javascript
/** Purpose: Redirects user back to individual article page with newly posted comment displayed */
app.post("/reader/article/:article_id",authorizeUser, (req, res) => {
    let article_id = req.params.article_id;
    let user_id = req.session.user_id;
    let sqlQuery = "INSERT INTO comments(article_id, user_id, comment_body, posted) VALUES(?, ?, ?, CURRENT_TIMESTAMP)";
    let details = [article_id, user_id, req.body.comment_body];
    db.run(sqlQuery, details, (err) => {
        if (err) {
            console.error(err.message);
            return res.status(500).send('Internal Server Error');
        }
        res.redirect(`/reader/article/${article_id}`);
    });
});
```

As mentioned in the previous sections, the article ID is retrieved from the route parameters. Similarly, this is used as an input parameter to create new comment records for a specific article.

Like function



```javascript
/** Purpose: Redirects user back to individual article page with like counter incremented */
app.post('/toggle-like/:article_id', authorizeUser, (req, res) => {
    const article_id = req.params.article_id;
    const updateQuery = `UPDATE articles SET likes = likes + 1 WHERE article_id = ?`;
    db.run(updateQuery, [article_id], (err) => {
        if (err) {
            console.error(err.message);
            return res.status(500).send('Internal Server Error');
        }
        res.redirect(`/reader/article/${article_id}`);
    });
});

/** Purpose: Redirects user back to individual article page with like counter decremented */
app.post('/toggle-dislike/:article_id', authorizeUser, (req, res) => {
    const article_id = req.params.article_id;
    const updateQuery = `UPDATE articles SET likes = likes - 1 WHERE article_id = ?`;
    db.run(updateQuery, [article_id], (err) => {
        if (err) {
            console.error(err.message);
            return res.status(500).send('Internal Server Error');
        }
        res.redirect(`/reader/article/${article_id}`);
    });
});
```

When the thumbs up or thumbs down buttons are toggled, the middleware performs the respective increment and decrement to the likes field in the articles table—for the specific article in which its article ID is retrieved from the route parameters.