

**IT 309 SOFTWARE ENGINEERING**

PROJECT DOCUMENTATION

Thread and Needle

Prepared by:

[Amina Nišić](mailto:amina.nisic@stu.ibu.edu.ba)

[Ema Kavazović-Devedžić](mailto:ema.kavazovic-devedzic@stu.ibu.edu.ba)

Proposed to:

**Nermina Durmić, Assist. Prof. Dr.**

**Aldin Kovačević, Teaching Assistant**

21.06.2023

TABLE OF CONTENTS

[1.1. About the Project](#_heading=h.30j0zll) 3

[1.2. Project Functionalities and Screenshots](#_heading=) 3

[**2. Project Structure**](#_heading=) **7**

[2.1. Technologies](#_heading=h.1fob9te) 7

[2.2. Database Entities](#_heading=) 8

[2.3. Design Patterns](#_heading=) 8

[2.4. Tests](#_heading=) 10

[**3. Conclusion**](#_heading=) **10**

## 1.1. About the Project

Thread and Needle is an inclusive web platform that serves as a lively hub for fashion design and sewing enthusiasts of all skill levels. With its user-friendly interface, it provides interactive tutorials, comprehensive resources, and a supportive community forum where users can exchange knowledge, seek advice, and showcase their creations. The platform's blog section further enriches the experience by featuring articles from experienced users, sharing insightful tips, inspiring designs, and in-depth tutorials to fuel creativity and foster a sense of belonging among the vibrant fashion and sewing community.

The website is available here: <https://thread-and-needle-kxe6.vercel.app/homepage>

It communicates with the back-end which is deployed here:

<https://thread-and-needle-543768a8777c.herokuapp.com/> (It will display an error only because it has nothing to fetch. Many backend functionalities, such as comments, can still be tested using Postman and this hostname.)

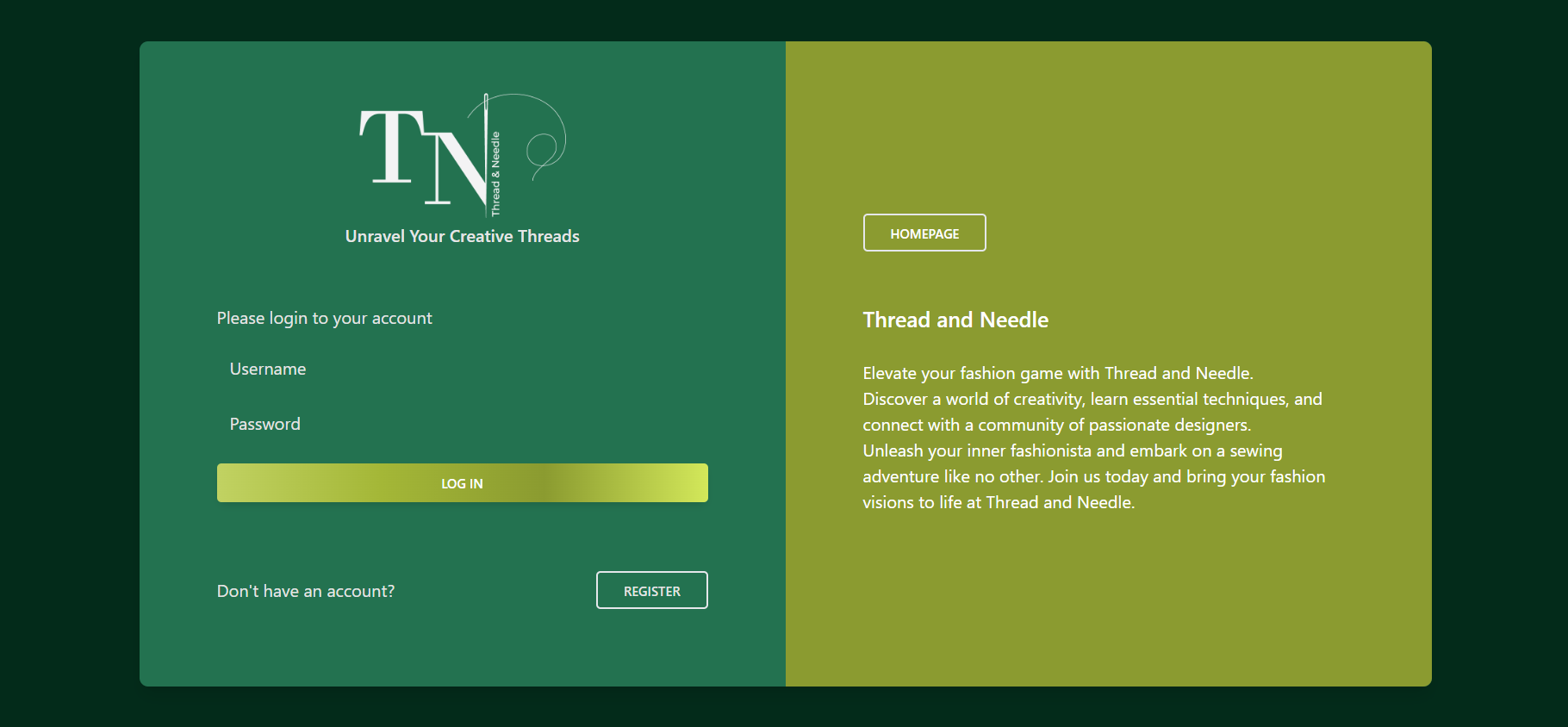
## 1.2. Project Functionalities and Screenshots

1. Account Creation:

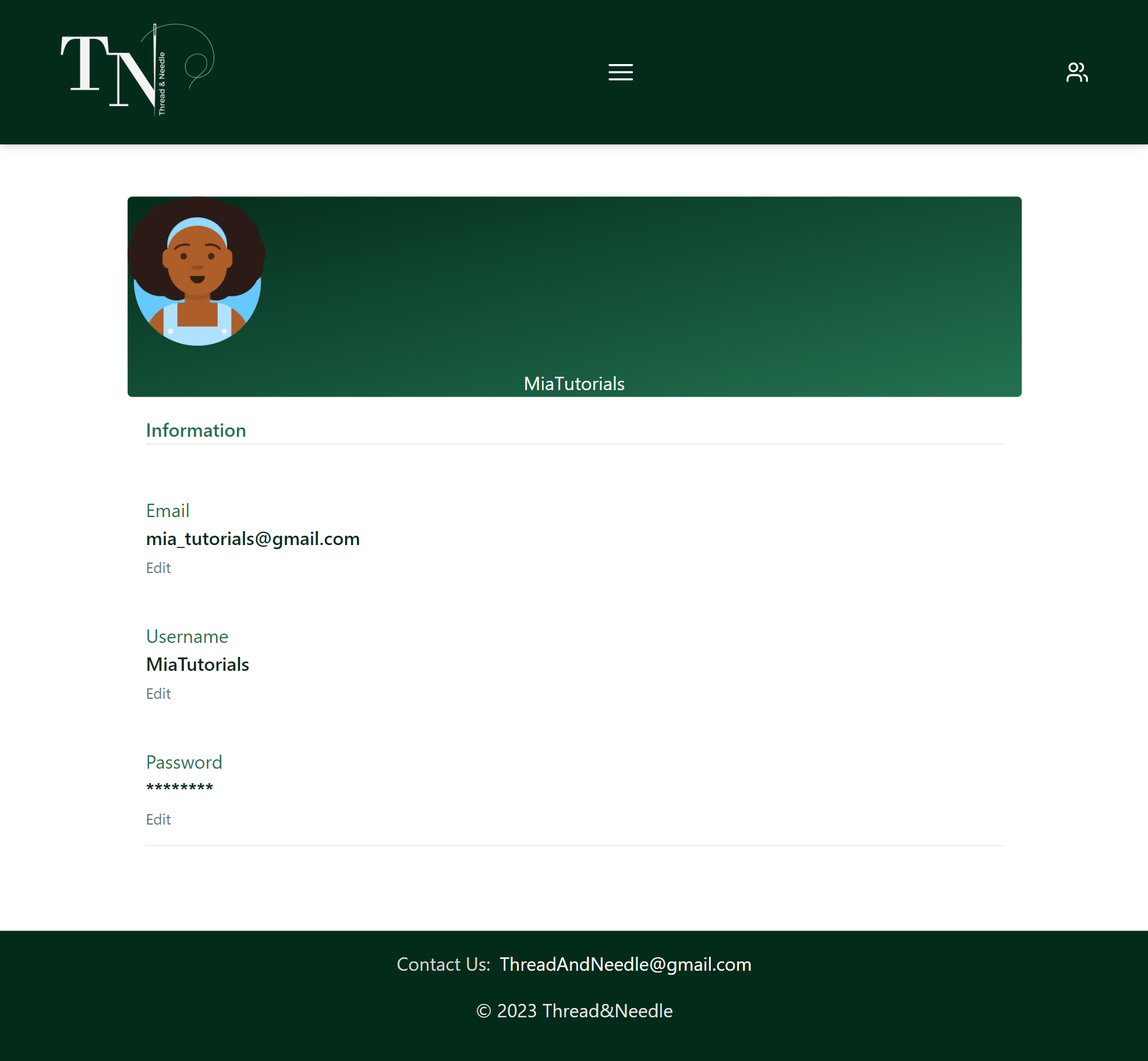
1. Registration



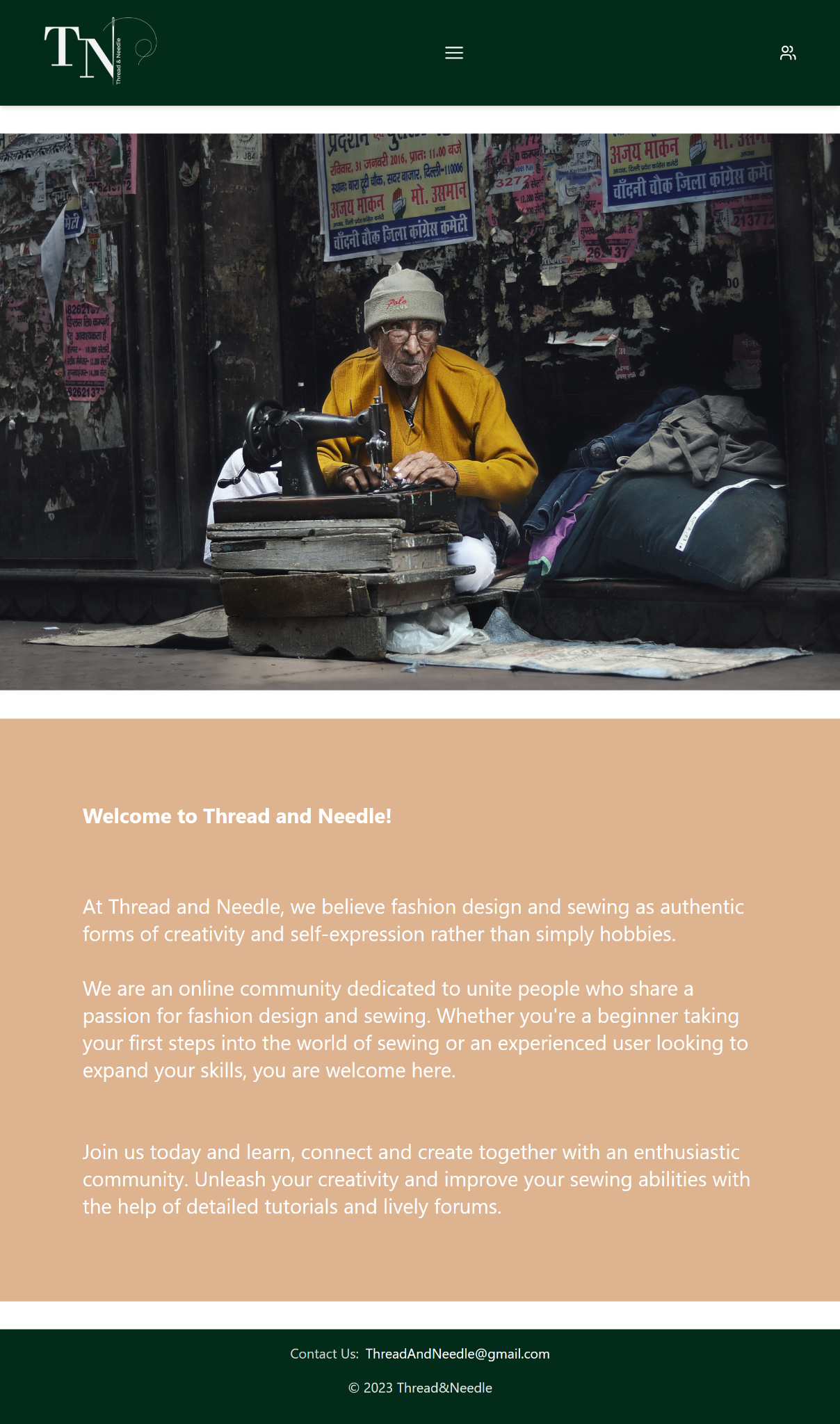
1. Login



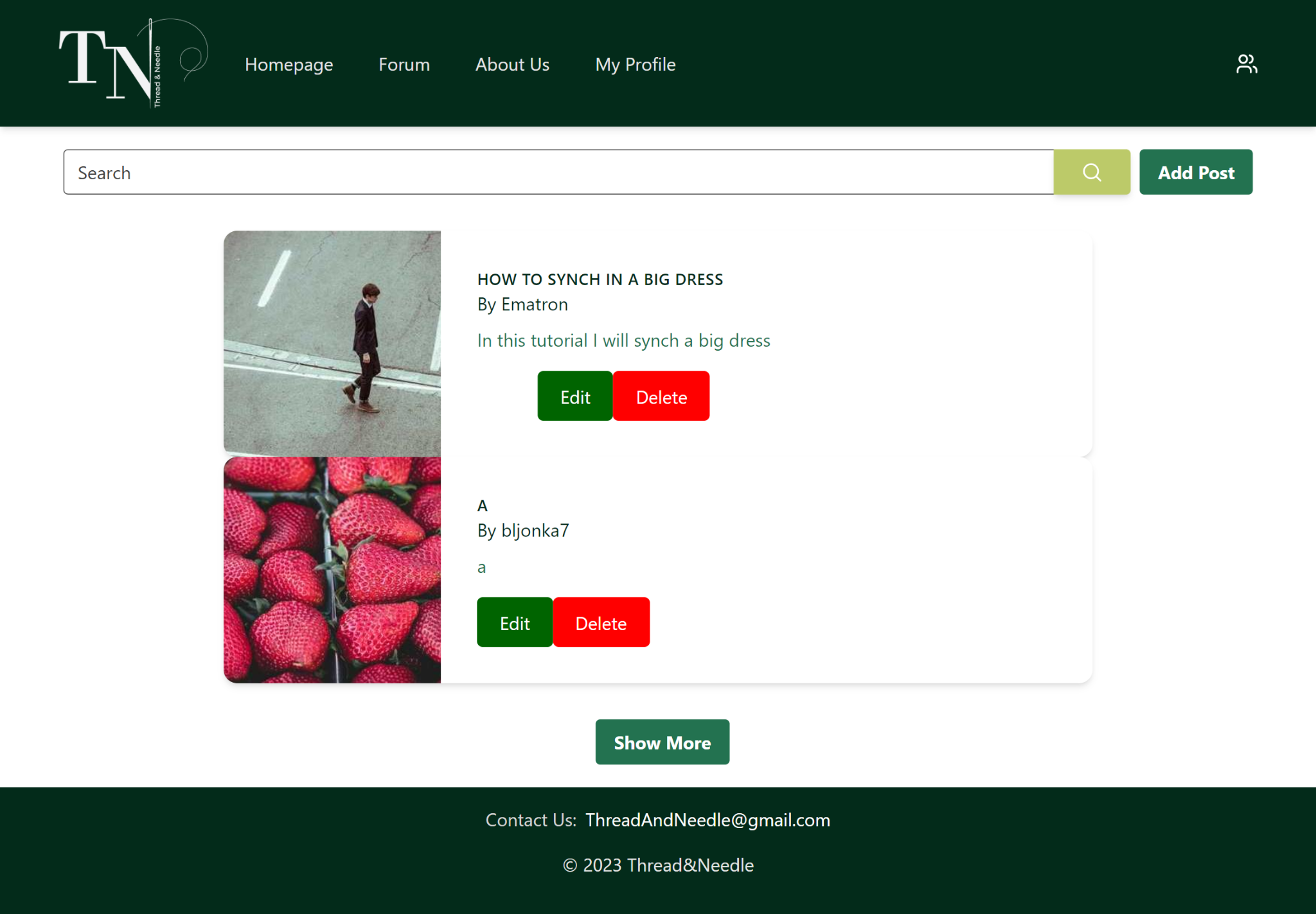
1. Edit Account/My Profile



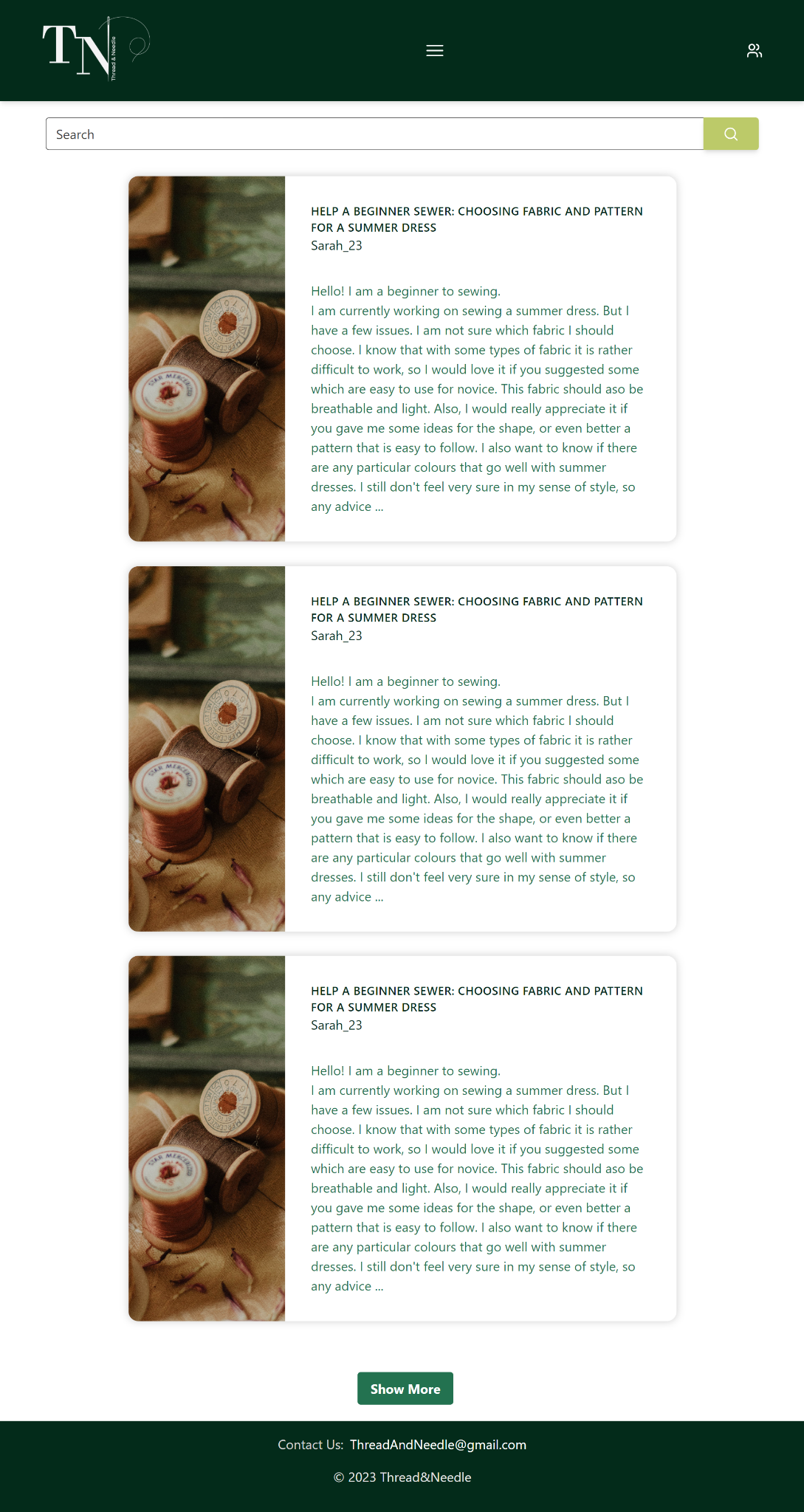
2. About Us



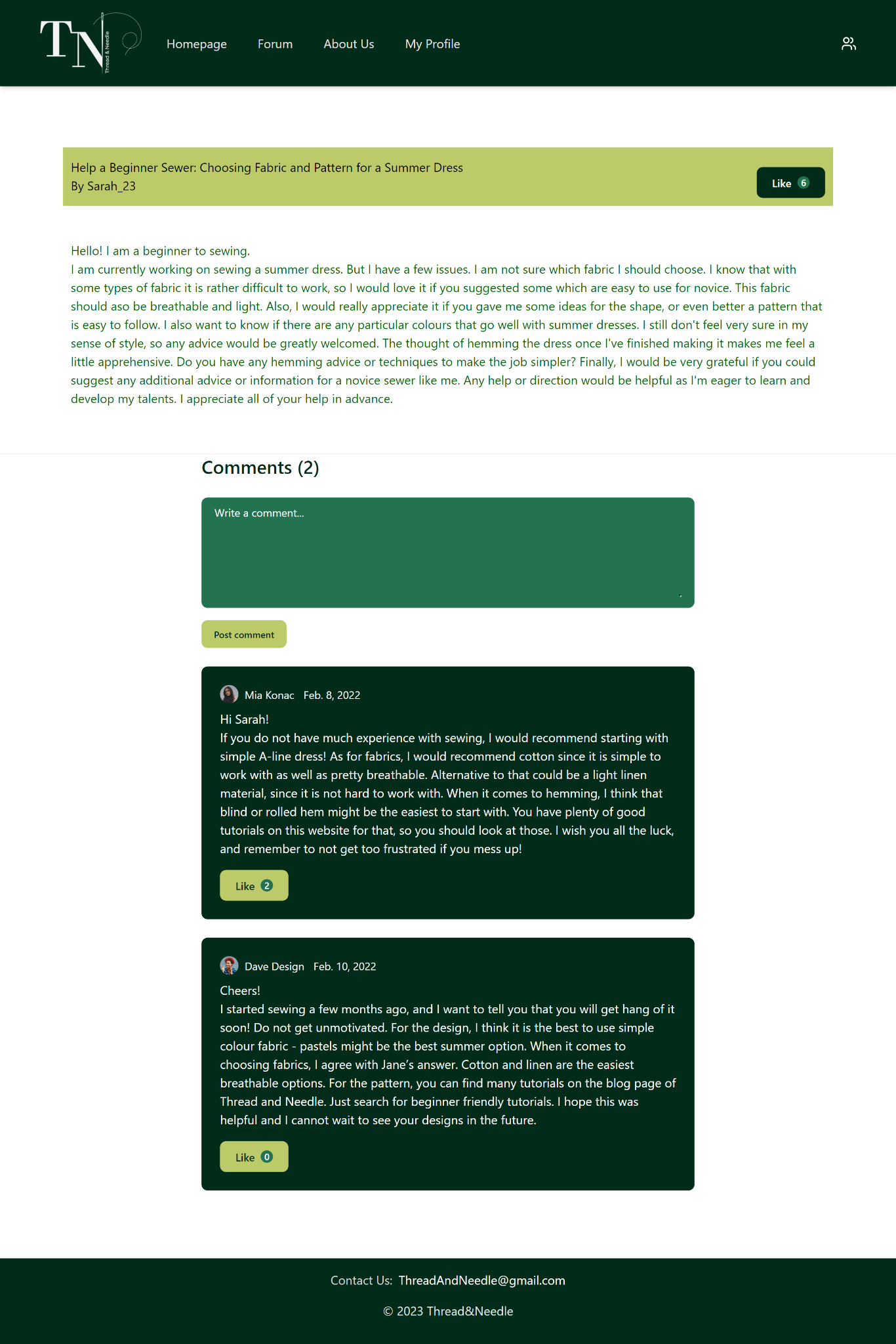
3. Homepage



5. Forum



6. Forum Post



# 2. Project Structure

## 2.1. Technologies

**Frontend:**

- We used Figma for designing and prototyping the user interface (UI) and user experience (UX).

- For building the actual frontend, we chose Angular. To keep things consistent, we followed the coding style guide provided in the AngularJS Style Guide (<https://github.com/toddmotto/angularjs-styleguide> ).

**Backend:**

- Our backend development was done using Node.js with the help of the Express web framework.

- We adopted the [Airbnb JavaScript Style Guide](https://github.com/airbnb/javascript) as our coding standard for the backend

**Database:**

- To manage our data, we used MySQL as the database system.

Currently, we have 3 functioning entities (They are hosted and the user can communicate with them using a UI that is easy to navigate)

The entities are:

-Users table (stores user data such as username, email, hashed password)

-Posts table (stores post id, title, description, blog text, user id and username of the person who posted)

-Forums table(stores forum ids, title and body of a forum post, as well as user id and username which is for now enabled to being null as comments are not yet implemented)

**Coding Standards:**

- We focused on making our code easy to understand and work with. This involved using clear and straightforward names, adding helpful documentation to the project, and creating components that are intuitive. We also implemented error handling to make it easier to identify and fix any issues that may arise.

In summary, we aimed to create a functional and user-friendly application by following established design and coding practices. We wanted our code to be easy to read, maintain, and troubleshoot.

## 2.2. Database Entities

* users
* posts
* forums
* …

## 2.3. Design Patterns

We have used one creational and one behavioral design pattern:

1. **Observer (behavioral design pattern)**

We're using the Observer pattern in the `blog-post-component.ts` file. This pattern helps components communicate with each other and ensures that our `BlogPostComponent` gets notified when posts are fetched.

In the file, we have a function called "subject" that serves as the subject or the entity being observed. It maintains a list of observers and notifies them when something important happens, such as the successful retrieval of posts.

By applying the Observer pattern, our code becomes more flexible and scalable. The subject component (in this case, `BlogPostComponent`) can inform all the registered observers, which might be other components interested in the fetched posts, without needing to know their specific identities. This decoupling allows for loose coupling between components and promotes better maintainability and extensibility.

To summarize, the Observer pattern enhances our codebase by facilitating effective communication between components, ensuring that our `BlogPostComponent` is notified when posts are fetched. It simplifies the process of adding new observers and promotes code separation and abstraction, making our code easier to maintain and extend.

1. **Singleton (creational design pattern)**

In the `registration-component.ts` file, we are using the Singleton pattern to ensure there is only one instance of the `ValidationService` class in our application.

The Singleton pattern serves several purposes in our implementation. Firstly, it guarantees consistent validation rules throughout the application. Regardless of where validation is performed, the same set of rules is consistently applied.

Additionally, the Singleton pattern simplifies access to the validation functionality from different parts of the application. By having a centralized access point to the `ValidationService` instance, we can easily utilize its validation capabilities without the need to pass instances around or create multiple instances.

Moreover, the pattern simplifies dependency management within the `registration-component.ts` file. We can directly access the `ValidationService` instance without explicitly instantiating it or passing it through the constructor. This simplification reduces code complexity and enhances maintainability.

Within the context of user registration, we leverage the `ValidationService` instance to validate the registration fields. When a user attempts to register but fails to fill in all the required fields, we display a frontend message stating "All fields are required". This ensures that users are prompted to fill in all necessary information for successful registration.

Additionally, we have incorporated regular expressions (regex) into the validation process. Specifically, we use a regex pattern to check the format of the email field. If an invalid email format is entered, we display the frontend message "Email format is not valid". This helps ensure that users provide a valid email address during registration.

## 2.4. Tests

We wrote Selenium Java tests. These tests are located in testings/src/TestCases.java.

In it, there is a class TestCases which includes test methods:

1. RegistrationTest (fill registration form and submit)
2. LoginTest (fill login form and submit)
3. CreatePostTest (navigate to create post, fill it and submit)
4. EditPostTest (navigate to edit post, modify content and save changes)
5. RedirectTest (test navigation by clicking different navigation links)

# 3. Conclusion

So far we are fairly satisfied with our work. We wish we could have done more and to have fully completed the website, however, considering our knowledge of the new frameworks we didn't have the agility to complete it faster. What we gained from this experience is that we had the opportunity to pick up two new frameworks with which we did not have any encounters yet and we are very pleased with our success of learning them since starting this project.

Having said that, the most challenging part was actually connecting the front-end to the back-end, as neither are difficult by themselves, but passing the right variables to communicate with the matching variables of the back-end proved to be a lot harder than we thought.

In conclusion, do not underestimate API calls.