

The Blog Challenge (TypeScript + CommonJS + Tailwind + Fetch)

Objective: Students will practice

- Dynamic template with EJS
- Express routing parameter
- lowerCase method of the Lodash
- Tailwind CSS (Play CDN) for UI styling
- Fetch API (POST) to update likes without page reload

Lab instruction

- There are 12 steps according to the blog challenge sheet posted on the channel.
- It is worth 28 points in total.
- Score criteria: full point (for output correct); -1 (for output does not correct); -1 (for not follow problem constraint)
- **Assignment Submission:**
 - Upload your solutions to CMU Mango assignments. The submission later than the 'due date' will get 50% off your score. At the 'close date', you cannot submit your assignment to the system.

A Blog Project

This project is to build a simple blog web application using **Express (TypeScript)**, **EJS**, and **Lodash**.

Users can create daily journal posts and **like** posts. The app includes:

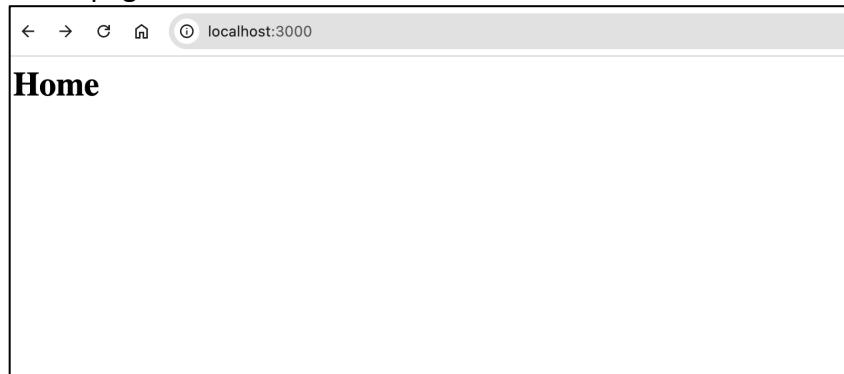
- View posts
- Compose a new daily journal post
- View the About and Contact pages
- Like a post (updates the like count)

TypeScript setup (Required)

- Install dependencies: `npm i express ejs lodash`
- Install dev dependencies: `npm i -D typescript nodemon ts-node @types/express @types/node @types/lodash`
- Create `tsconfig.json`: `npx tsc --init`
- `tsconfig.json`: `"module": "commonjs", "rootDir": "./src", "outDir": "./dist"`
- `package.json` scripts: `dev/build/start`
- run: `npm run dev`

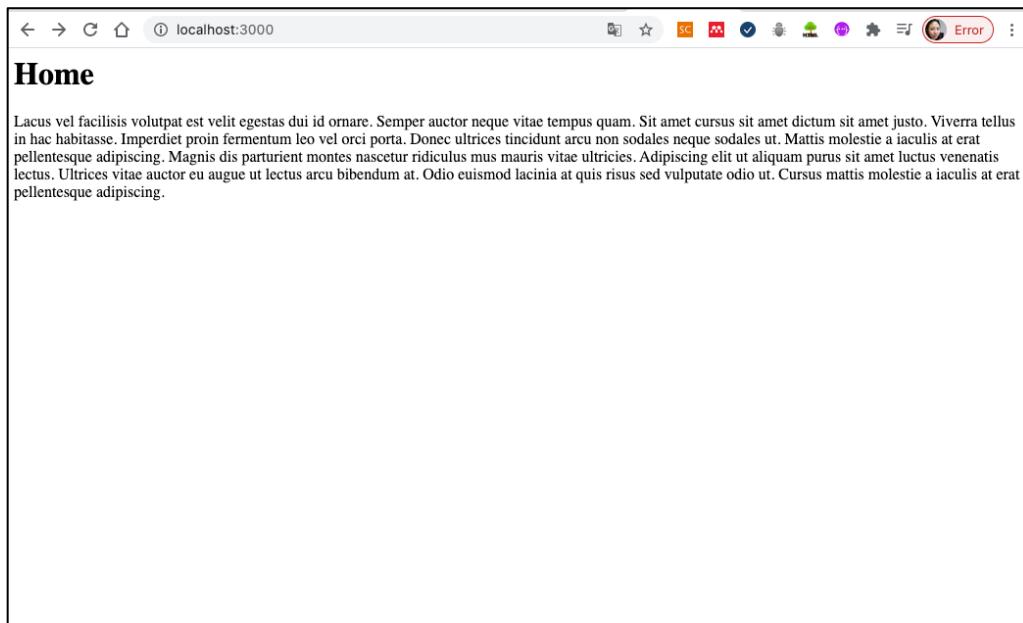
Step 0: Set up the Blog project (1 point)

1. Download the starting files from the Blog-resource file. In the folder, you will find `src/app.ts` and two folders (public and views).
2. Add the '`<h1>Home</h1>`' inside **views/home.ejs**.
3. Create a root (/) route GET to render the `home.ejs` inside `src/app.ts`. Use `res.render()`.
4. Run the server with `npm run dev` and test on your browser (`http://localhost:3000/`). You should see the Home with `<h1>` rendered on the home page.



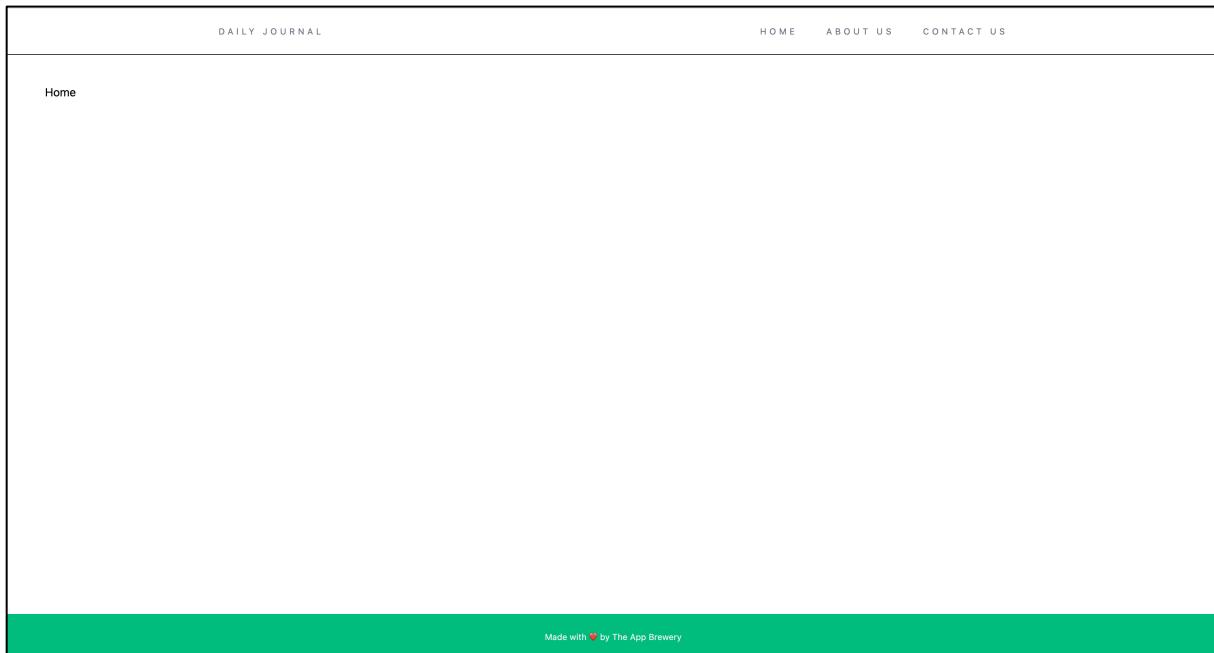
Step 1: Add the `homeStartingContent` to `home.ejs` (2 points)

1. Modify the `res.render()` you have created in Step 0 to inject the constant `homeStartingContent` into the home page inside the paragraph tag. Note that the constant `homeStartingContent` is already defined in `src/app.ts`.
2. Use `<%= ... %>` inside the `home.ejs` to output the `homeStartingContent` value
3. The expected output is as follows when you run <http://localhost:3000/> on the browser.

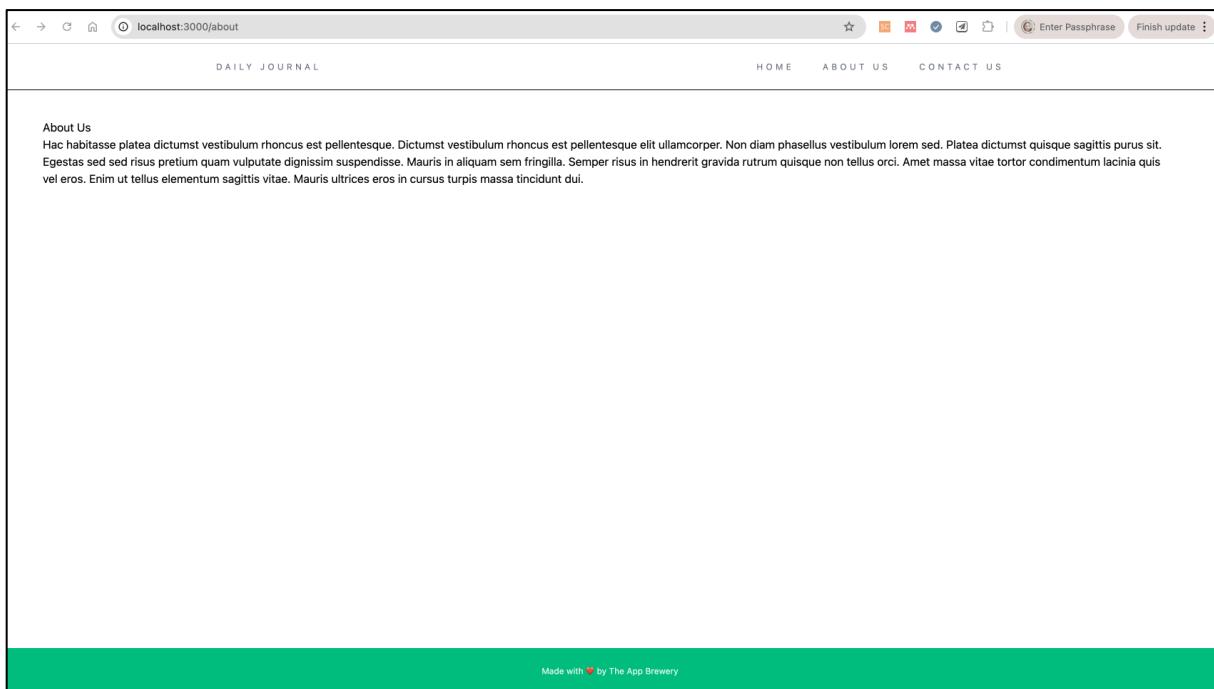
**Step 2: Modify the home template with its partials (2 points)**

Currently, the home page contains only `<h1>` and `<p>` tags. The other parts' codes of the home page are in the `header.ejs` and `footer.ejs`. So, you need to replace the header and footer with the ejs partials and include them into the `home.ejs`.

1. Create a new subfolder named **partials** inside the view folder.
2. Move **header.ejs** and **footer.ejs** into **partials** folder.
3. Use `<%- ... %>` to include the 'partials/header' and 'partials/footer' into the `home.ejs`.
4. The expected output is as follows when you run <http://localhost:3000/> on the browser.

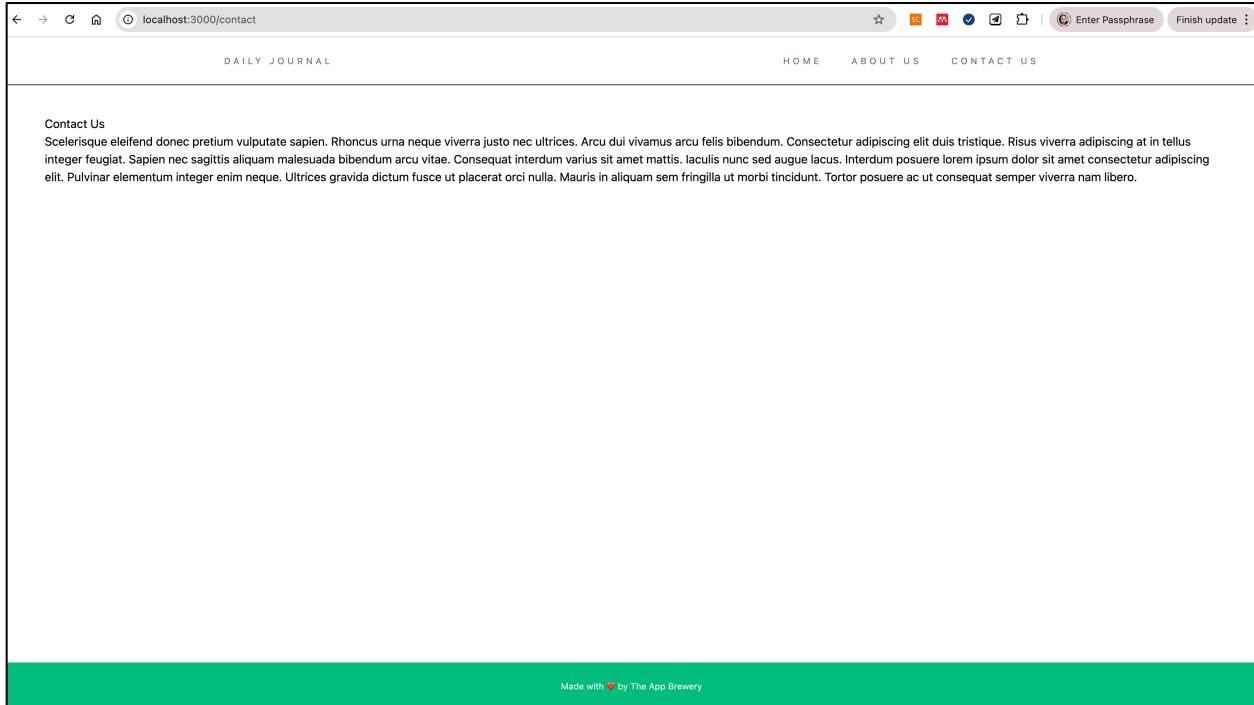
**Step 3: Create the About Us page with the EJS template (2 points)**

1. Create a '/about' route GET to render the about.ejs inside src/app.ts.
2. Inject the constant **aboutContent** to about.ejs
3. Create the about.ejs with its partials header and footer.
4. The expected output is as follows when you run <http://localhost:3000/about> on the browser AND when you click the menu 'ABOUT US' on the home page.

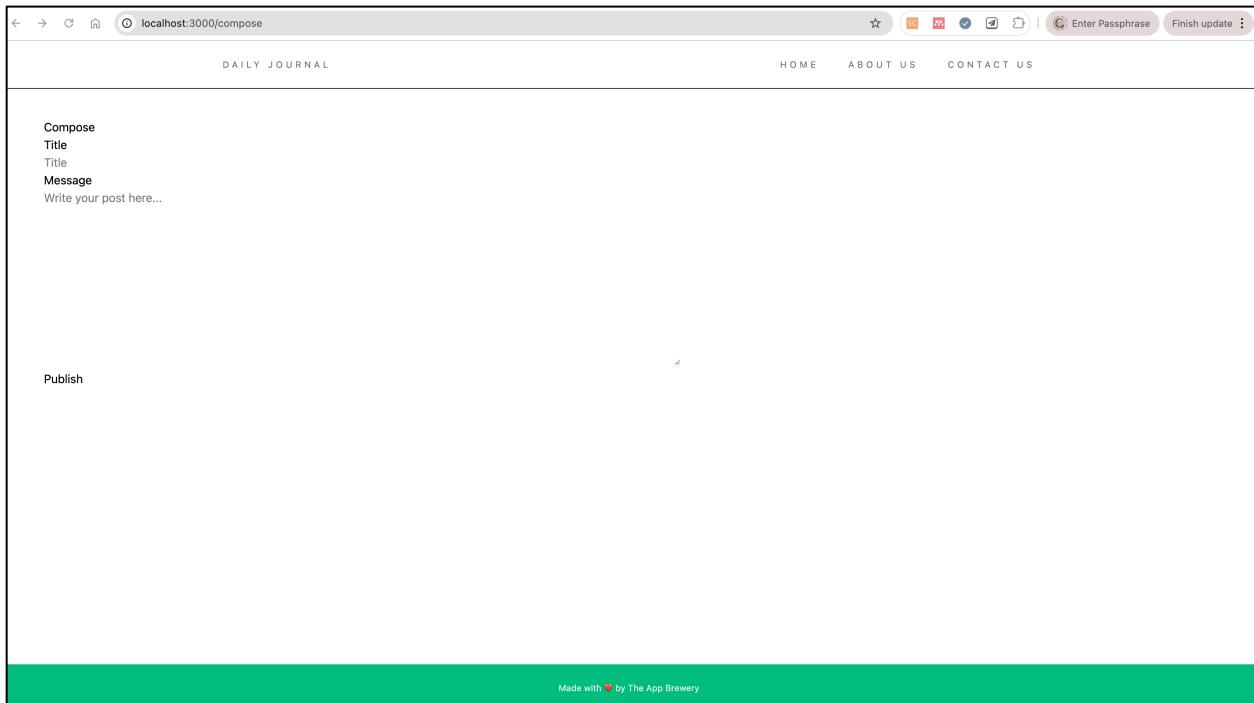


Step 4: Create the Contact Us page with the EJS template (2 points)

1. Create '/contact' route GET to render the contact.ejs inside src/app.ts.
2. Inject the constant **contactContent** to contact.ejs
3. Create the contact.ejs with its partials header and footer.
4. The expected output is as follows when you run <http://localhost:3000/contact> on the browser AND when you click the menu 'CONTACT US' on the home page.

**Step 5: Create the Compose page with the EJS template (1 point)**

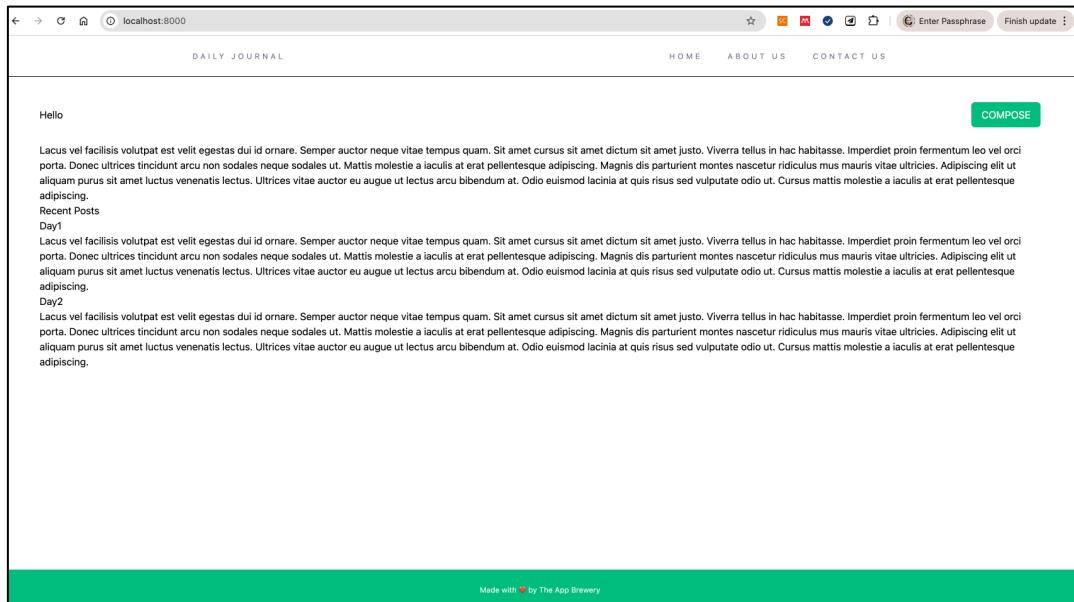
1. Create a '/compose' route GET to render the compose.ejs inside src/app.ts.
2. Create the compose.ejs with its partials header and footer.
3. The expected output is as follows when you run <http://localhost:3000/compose> on the browser.



For POST form data, make sure src/app.ts has: `app.use(express.urlencoded({ extended: true }));`

Step 6: Create a new post to show on the home page (3 points)

1. Declare an empty Array of global variable **posts**. The posts will be used to store the object post which has 2 keys title and content.
2. Create a '**/compose**' route **POST** to send **postTitle** and **postBody** from the Compose form to be stored in the Array **posts**. (Hints: In the callback function, keep the postTitle and postBody in an object and add this object to the Array posts.)
3. Log the variable posts to see a result.
4. Add `res.redirect('/')` as the posts will be displayed on the home page after the user submitted the post.
5. Then go back to `app.get('/')` to modify the `res.render()` to inject the **posts** into the **home.ejs**.
6. Finally, modify the **home.ejs** to render the injected posts to be displayed on the home page. The following is the expected output of 2 posts.



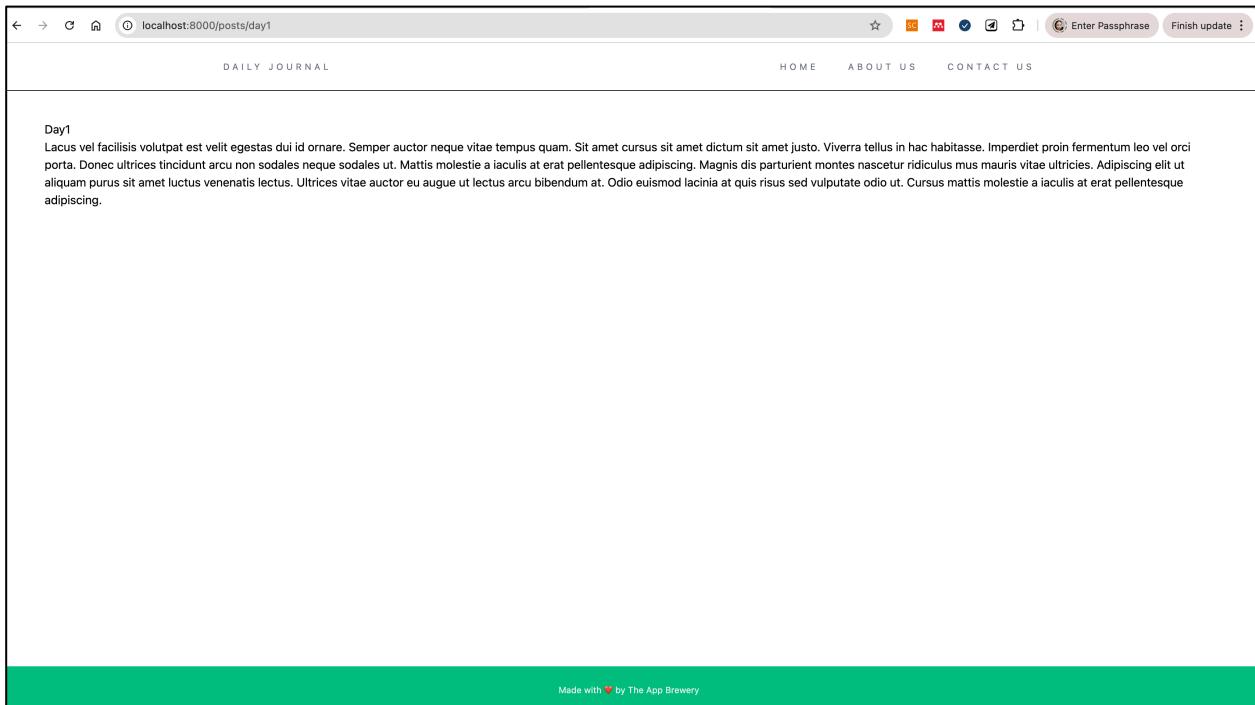
7. You can get a random text from <https://www.lipsum.com/>

Step 7: Working with Express Routing Parameter (3 points)

1. Study how Express Routing Parameter works on <http://expressjs.com/en/guide/routing.html#route-parameters>
2. Add a '/posts/:postName' route GET inside src/app.ts. In the callback function, log the req.params.postName to see the postName the user entering on the URL path.
3. Test the callback function. The following is the expected output in the console when the user enters <http://localhost:3000/posts/Day1> on the browser.

```
Server started on port 3000
Day1
[]
```

8. Then, display the selected post on the **post.ejs** by adding the **res.render()** to inject **the selected post title** and **the selected post content** into the **post.ejs**. The expected output is as follows.



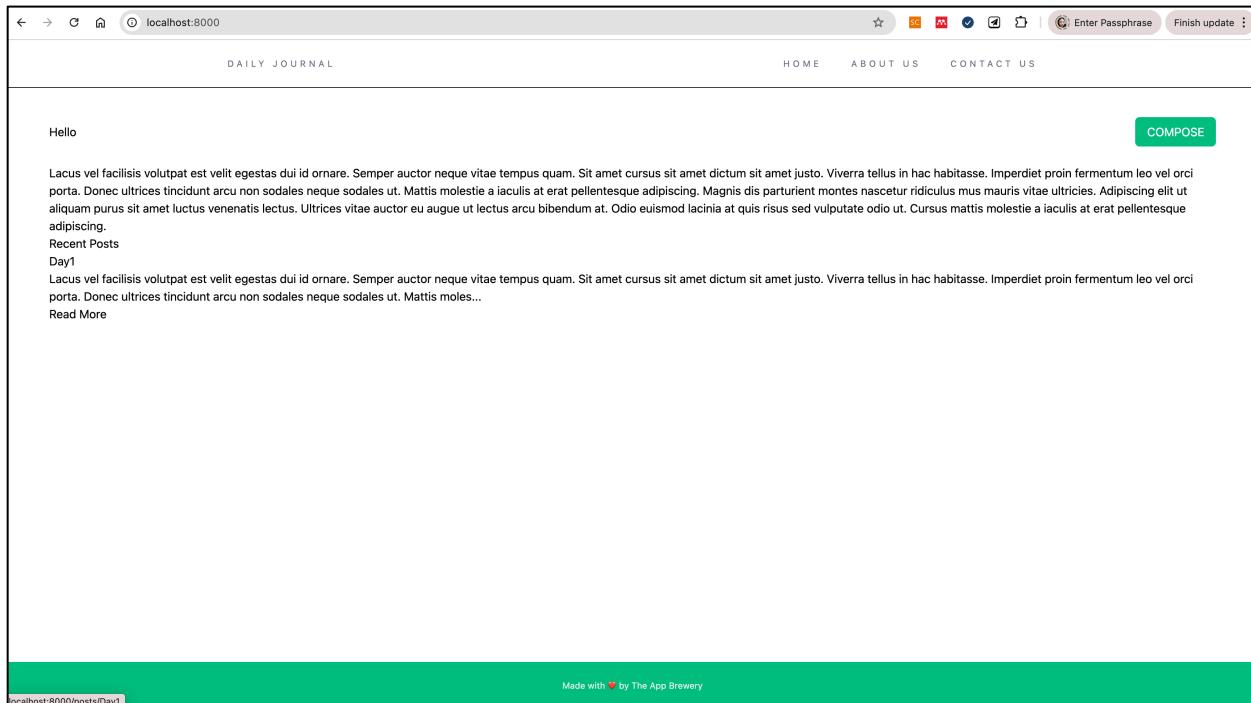
Step 8: Work with Lodash module (2 points)

As it is usually the user will type all the lower case letters on the URL, so you must do something before matching them.

1. Study how **lowerCase** method of **Lodash module** works on <https://lodash.com/docs/4.17.15#lowerCase>
2. In src/app.ts, import Lodash (TypeScript):
3. Modify your code in Step 7 that uses lowerCase method of Lodash module to convert the **postName on the URL** and the **postName** stored in the Array **posts** to be all lower case letters before matching them.
4. The expected output is as follows when you type <http://localhost:3000/posts/day1> or <http://localhost:3000/posts/Day 1>

Step 10: Complete the Blog (3 points)

Modify the **home.ejs** to display each post with **300 characters** and an archer link for read more detail of the selected post rendered in the **post.ejs**. The expected outputs are as follows.



localhost:8000

DAILY JOURNAL

HOME ABOUT US CONTACT US

Hello

COMPOSE

Lacus vel facilisis volutpat est velit egestas dui id ornare. Semper auctor neque vitae tempus quam. Sit amet cursus sit amet dictum sit amet justo. Viverra tellus in hac habitasse. Imperdiet proin fermentum leo vel orci porta. Donec ultrices tincidunt arcu non sodales neque sodales ut. Mattis molestie a iaculis at erat pellentesque adipiscing. Magnis dis parturient montes nascetur ridiculus mus mauris vitae ultricies. Adipiscing elit ut aliquam purus sit amet luctus venenatis lectus. Ultrices vitae auctor eu augue ut lectus arcu bibendum at. Odio euismod lacinia at quis risus sed vulputate odio ut. Cursus mattis molestie a iaculis at erat pellentesque adipiscing.

Recent Posts

Day1

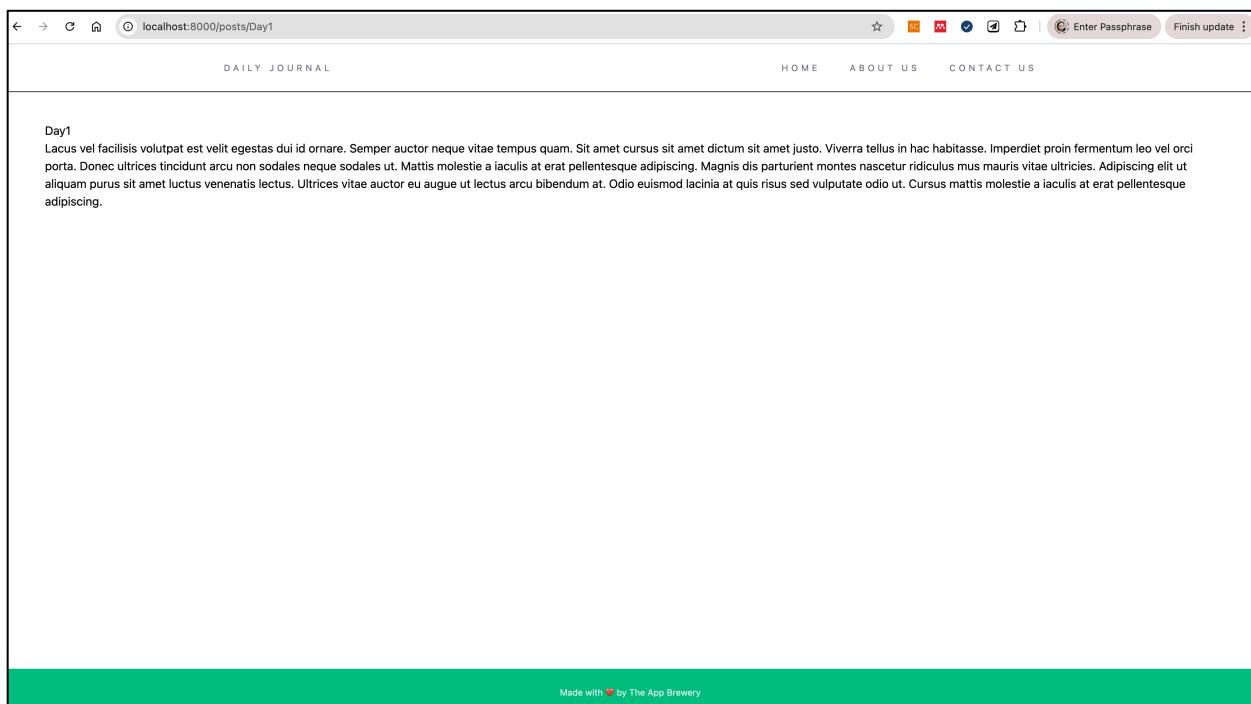
Lacus vel facilisis volutpat est velit egestas dui id ornare. Semper auctor neque vitae tempus quam. Sit amet cursus sit amet dictum sit amet justo. Viverra tellus in hac habitasse. Imperdiet proin fermentum leo vel orci porta. Donec ultrices tincidunt arcu non sodales neque sodales ut. Mattis moles...

Read More

Made with ❤ by The App Brewery

localhost:8000/posts/Day1

When the user clicks on the '**Read More**' link of post Day1, the post of Day 1 is loaded.



localhost:8000/posts/Day1

DAILY JOURNAL

HOME ABOUT US CONTACT US

Day1

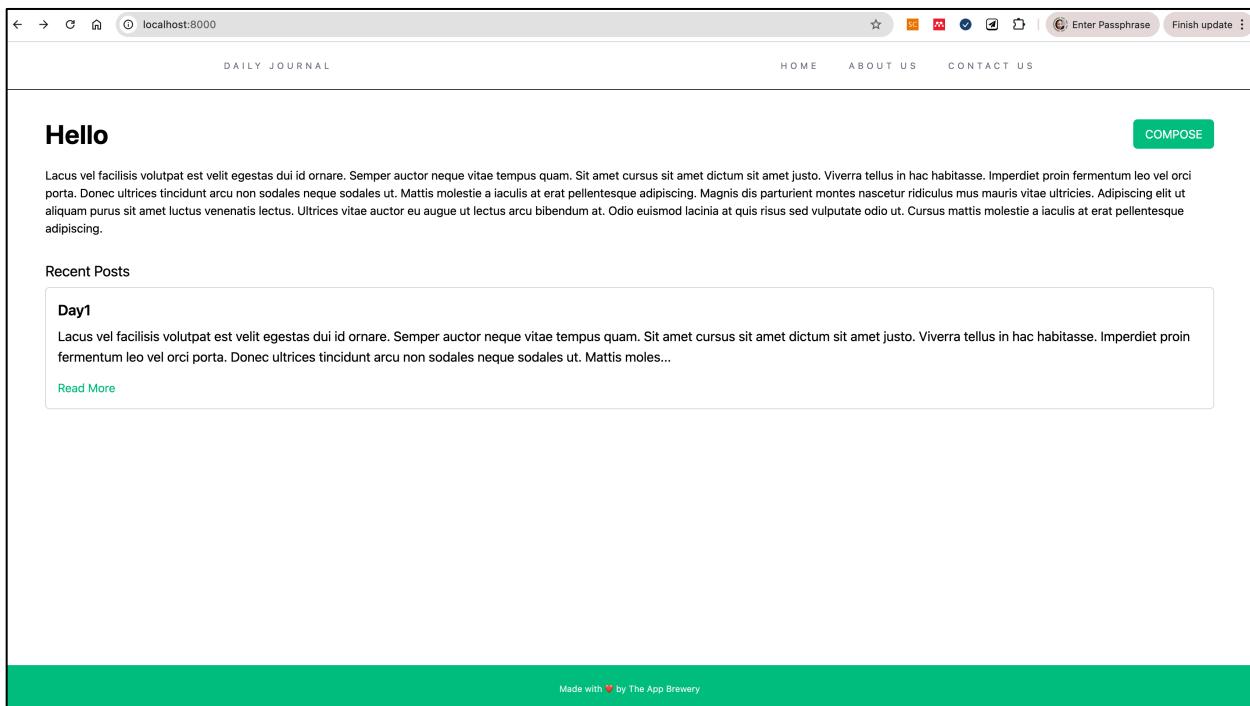
Lacus vel facilisis volutpat est velit egestas dui id ornare. Semper auctor neque vitae tempus quam. Sit amet cursus sit amet dictum sit amet justo. Viverra tellus in hac habitasse. Imperdiet proin fermentum leo vel orci porta. Donec ultrices tincidunt arcu non sodales neque sodales ut. Mattis molestie a iaculis at erat pellentesque adipiscing. Magnis dis parturient montes nascetur ridiculus mus mauris vitae ultricies. Adipiscing elit ut aliquam purus sit amet luctus venenatis lectus. Ultrices vitae auctor eu augue ut lectus arcu bibendum at. Odio euismod lacinia at quis risus sed vulputate odio ut. Cursus mattis molestie a iaculis at erat pellentesque adipiscing.

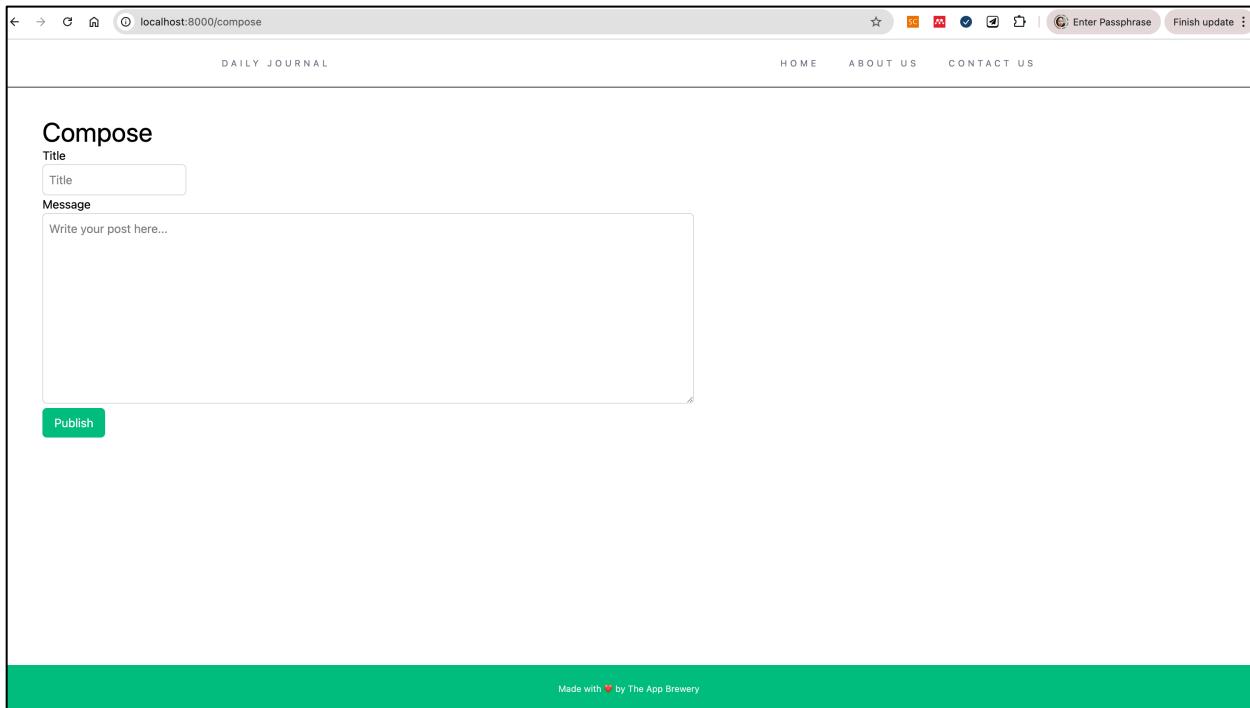
Made with ❤ by The App Brewery

localhost:8000/posts/Day1

Step 11: Apply Tailwind CSS (Play CDN) (3 points)

- Add Tailwind Play CDN in your header partial (views/partials/header.ejs):
`<script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>`
- Minimum UI requirements:
 - Home page: posts displayed as cards, each card shows title, preview text, Read More link, likes count, and a Like button.
 - Compose page: inputs and submit button are styled (spacing, borders, hover).
 - Post detail page: clear title + readable body text.
 - Header and footer appear on every page and spacing is consistent (use the same container width).
- Example of UI is below.





Step 12: Like Button with Fetch POST (4 points)

Goal: Click Like on a post card and update the likes count immediately (no page reload).

A) Update your post data (TypeScript)

Make sure each post has: id (unique) and likes (number). If your starter code already declares likes/id in the Post type, you only need to set values when creating a new post.

Recommended:

```
const newPost = {
  id: Date.now().toString(),
  title: postTitle,
  body: postBody,    // or content
  likes: 0,
};
```

B) Create an API endpoint (server)

Make sure JSON middleware exists (starter code already has this):

```
app.use(express.urlencoded({ extended: true })); // form data
app.use(express.static("public"));           // /public files
app.use(express.json());                    // JSON for fetch()
```

Create this route in src/app.ts:

```
app.post("/api/posts/:id/like", (req, res) => {
  const { id } = req.params;
  const post = posts.find(p => p.id === id);

  if (!post) {
    return res.status(404).json({ error: "Post not found" });
  }

  post.likes += 1;
  return res.json({ id: post.id, likes: post.likes });
});
```

C) Update Home page UI (home.ejs)

For each post card, add:

A likes text element with id="likes-<%= post.id %>"

A Like button with class="like-btn" and data-id="<%= post.id %>"

An optional status element with id="status-<%= post.id %>"

```
<p id="likes-<%= post.id %>">Likes: <%= post.likes %></p>
<button class="like-btn" data-id="<%= post.id %>">Like</button>
<p id="status-<%= post.id %>" class="text-sm text-gray-500"></p>
```

D) Add client-side JS using fetch()

Create public/like.js and include it at the bottom of home.ejs:

```
<script src="/like.js"></script>
```

In public/like.js implement:

Add click listeners for all elements with class "like-btn".

Send POST request: fetch(`/api/posts/\${id}/like`, { method: "POST" })

Check res.ok then read res.json()

Update only the likes text in the DOM (no reload).

UI states: disable the clicked button while loading; show "Liking..." and show error text if failed.

Important note: fetch() only rejects on network errors — you must check res.ok for 404/500.

Here is the example of the result.

The screenshot shows a web application interface for a 'DAILY JOURNAL'. The top navigation bar includes links for 'HOME', 'ABOUT US', and 'CONTACT US', along with a 'COMPOSE' button and a 'Finish update' link. The main content area displays two journal entries. Each entry has a title ('Day1' or 'Day2'), a preview text, a 'Read More' button, and a 'Like' button. The 'Day1' entry shows 'Like: 2', while the 'Day2' entry shows 'Like: 0'. At the bottom of the page, there is a green footer bar with the text 'Made with ❤ by The App Brewery'.

How to Submit

Submit your assignment on CMU Mango.

Upload ONE zip file named: Blog_StudentID.zip

Include:

- src/ (TypeScript source code)
- views/ (EJS templates, including partials)
- public/ (static files such as like.js, images, etc.)
- package.json, tsconfig.json

Do NOT include: node_modules/

Also include 2 screenshots (jpg/png):

- Home page showing Tailwind UI with at least 2 posts

- Like button clicked and likes count updated without reloading the page