Project Structure

This is the structure of the project.

```
├── public
│   ├── script.js
│   └── style.css
├── models
│   ├── post.js
│   └── user.js
├── views
│   ├── login.ejs
│   ├── main.ejs
│   ├── newpost.ejs
│   └── signup.ejs
├── app.js
├── createAdmin.js
└── .env
```

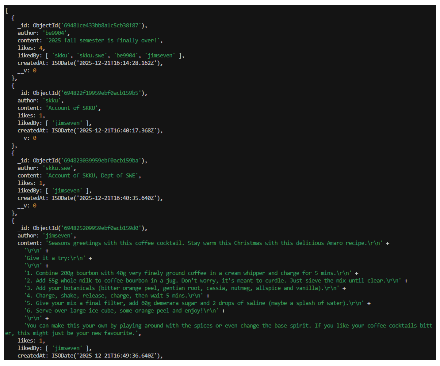
Models

1. post.js

This file defines the post model. It is the schema for all user-generated posts stored in the MongoDB database.

The schema specifies five key fields required for the posting and social interaction functionality.

- **author** (String, Required): This field stores the unique identifier of the user who created the post. It is marked as required to ensure no anonymous posts can be created.
- **content** (String, Required): This holds the text contents of the post. It is also mandatory to prevent empty posts being sent to the database.
- **likes** (Number, Default: `0`): This field serves as a cached counter for the total number of likes. It is saved as a separate field to allow faster read operations when rendering the main feed because the database does not need to compute array lengths for every post.
- **createdAt** (Date, Default: Date.now): This field automatically records the time when the post was created.
- **likedBy** (Array of Strings, Default: `[]`): This array stores the ids of all users who have liked the post. It is defined to prevent duplicate likes from the same user. Before registering a like, the server checks if the user's id exists in this array.



```
[
  {
    _id: ObjectId('69481ce433bb8a1c5cb38f87'),
    author: 'be9904',
    content: '2025 fall semester is finally over!',
    likes: 4,
    likedBy: [ 'skku', 'skku.swe', 'be9904', 'jimseven' ],
    createdAt: ISODate('2025-12-21T16:14:28.162Z'),
    __v: 0
  },
  {
    _id: ObjectId('694822f19959ebf0acb159b5'),
    author: 'skku',
    content: 'Account of SKKU',
    likes: 1,
    likedBy: [ 'jimseven' ],
    createdAt: ISODate('2025-12-21T16:40:17.368Z'),
    __v: 0
  },
  {
    _id: ObjectId('694823039959ebf0acb159ba'),
    author: 'skku.swe',
    content: 'Account of SKKU, Dept of SWE',
    likes: 1,
    likedBy: [ 'jimseven' ],
    createdAt: ISODate('2025-12-21T16:40:35.640Z'),
    __v: 0
  },
  {
    _id: ObjectId('694825209959ebf0acb159d0'),
    author: 'jimseven',
    content: 'Seasons greetings with this coffee cocktail. Stay warm this Christmas with this delicious Amaro recipe.\r\n' +
      '\r\n' +
      'Give it a try:\r\n' +
      '\r\n' +
      '1. Combine 200g bourbon with 40g very finely ground coffee in a cream whipper and charge for 5 mins.\r\n' +
      '2. Add 55g whole milk to coffee-bourbon in a jug. Don’t worry, it’s meant to curdle. Just sieve the mix until clear.\r\n' +
      '3. Add your botanicals (bitter orange peel, gentian root, cassia, nutmeg, allspice and vanilla).\r\n' +
      '4. Charge, shake, release, charge, then wait 5 mins.\r\n' +
      '5. Give your mix a final filter, add 60g demerara sugar and 2 drops of saline (maybe a splash of water).\r\n' +
      '6. Serve over large ice cube, some orange peel and enjoy!\r\n' +
      '\r\n' +
      'You can make this your own by playing around with the spices or even change the base spirit. If you like your coffee cocktails bitt
er, this might just be your new favourite.',
    likes: 1,
    likedBy: [ 'jimseven' ],
    createdAt: ISODate('2025-12-21T16:49:36.640Z'),
```

Posts saved in DB

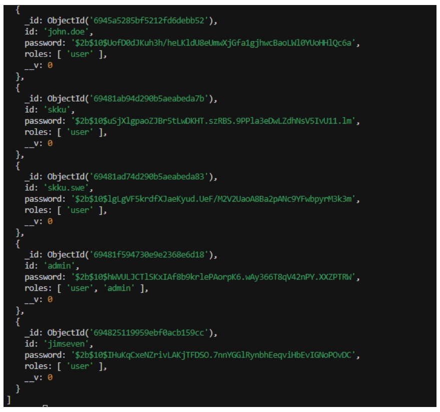
The schema is compiled into a Mongoose Model via `mongoose.model('Post', postSchema)` and exported.

2. user.js

This file defines the user model. It is the schema for all registered users stored in the MongoDB database, handling authentication and authorization data.

The schema specifies three key fields required for user management.

- **id** (String, Required, Unique): This field serves as the unique username for the user. It is marked as unique to prevent multiple users from registering with the same identifier, ensuring reliable account retrieval.
- **password** (String, Required): This stores the user's password. It is intended to hold a hashed string (using bcrypt) rather than plain text, ensuring security for user credentials.
- **roles** (Array of Strings, Default: `['user']`): This array defines the permissions associated with the account (e.g., 'user', 'admin'). It allows for role-based access control, defaulting to a standard 'user' role upon creation.



```
  {
    _id: ObjectId('6945a5285bf5212fd6debb52'),
    id: 'john.doe',
    password: '$2b$10$UofD0dJKuh3h/heLKldU8eUmwXjGfa1gjhwcBaoLWl0YUoHHlQc6a',
    roles: [ 'user' ],
    __v: 0
  },
  {
    _id: ObjectId('69481ab94d290b5aeabeda7b'),
    id: 'skku',
    password: '$2b$10$uSjXlgpaoZJBr5tLwDKHT.szRBS.9PPla3eDwLZdhNsV5IvU11.lm',
    roles: [ 'user' ],
    __v: 0
  },
  {
    _id: ObjectId('69481ad74d290b5aeabeda83'),
    id: 'skku.swe',
    password: '$2b$10$lgLgVF5krdfXJaeKyud.UeF/M2V2UaoA8Ba2pANc9YFwbpyrM3k3m',
    roles: [ 'user' ],
    __v: 0
  },
  {
    _id: ObjectId('69481f594730e9e2368e6d18'),
    id: 'admin',
    password: '$2b$10$hWVULJCTlSKxIAf8b9krlePAorpK6.wAy366T8qV42nPY.XXZPTRW',
    roles: [ 'user', 'admin' ],
    __v: 0
  },
  {
    _id: ObjectId('694825119959ebf0acb159cc'),
    id: 'jimseven',
    password: '$2b$10$IHuKqCxeNZrivLAKjTFDSO.7nnYGGlRynbhEeqviHbEvIGNoPOvDC',
    roles: [ 'user' ],
    __v: 0
  }
]
```

Users saved in DB

The schema is compiled into a Mongoose Model via `mongoose.model('User', userSchema)` and exported.

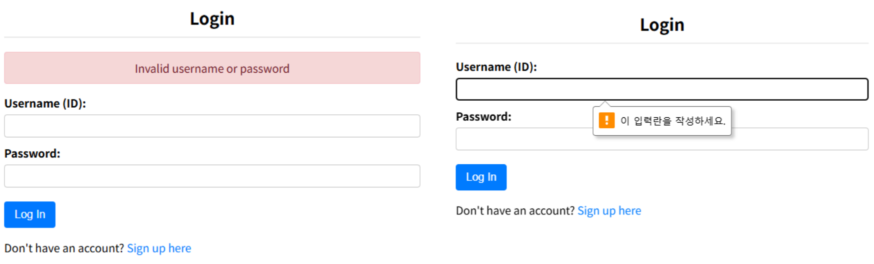
Views

1. login.ejs

This file defines the login view. It serves as the frontend interface for user authentication, rendered by the server using the EJS template.

The file contains three key structural components required for the login functionality.

- **Error Handling Block** (EJS condition): This section checks if an error variable was passed from the server. If present (e.g., "Invalid password"), it dynamically renders a div with the class error to display the message to the user.
- **Login Form** (HTML form): This form collects user credentials. It is configured with method="POST" and action="/login", ensuring that when the login button is clicked, the data is securely sent to the server's login route for verification.
- **Input Fields** (HTML inputs): These are the required text fields for the id and password. The name attributes (username, password) must match the variable names expected by the backend controller to process the request correctly.



Login

Invalid username or password

Username (ID):

Password:

Log In

Don't have an account? Sign up here

Login

Username (ID):

이 입력란을 작성하세요.

Password:

Log In

Don't have an account? Sign up here

Login Fail Cases

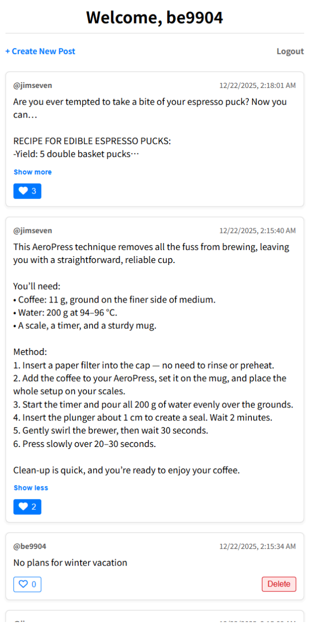
The file concludes with a navigation link redirecting unregistered users to the signup page.

2. main.ejs

This file defines the main feed view. It serves as the core user interface for the application, displaying the stream of user-generated content.

The file contains four key structural components required for the feed functionality.

- **Library Imports** (HTML head): This section loads external resources for the page's appearance and behavior. It includes the custom stylesheet (/style.css), the Font Awesome library for scalable vector icons (used in the like button), and the client-side logic script (/script.js) loaded with the defer attribute to ensure the HTML parses fully before execution.
- **Navigation Bar** (HTML div): This container provides user controls. It includes links to create a new post and to log out, ensuring the user can easily manage their session and content creation.
- **Feed Loop** (EJS loop): This block iterates through the posts array passed from the server. It dynamically generates HTML for each post, formatting fields such as the author's name, the creation date, and the post contents. It handles the empty state case by displaying a message if the database returns no posts.
- **Interactive Post Actions** (Conditional Logic): This section manages user interactions within each post card.
  - **Content Collapsing**: Calculates whether a post is too long (based on character count or newlines) and conditionally renders a show more button to toggle visibility.
  - **Like Button**: Renders a dynamic button that checks if the current user's id is in the likedBy array, setting the initial heart icon style (solid or outline) accordingly.
  - **Delete Button**: Uses a conditional check to render a delete form only if the current user is the post's author or holds an 'admin' role.



Welcome, be9904

+ Create New Post　Logout

@jimseven　12/22/2025, 2:18:01 AM

Are you ever tempted to take a bite of your espresso puck? Now you can…

RECIPE FOR EDIBLE ESPRESSO PUCKS:
-Yield: 5 double basket pucks⋯

Show more

♥ 3

@jimseven　12/22/2025, 2:15:40 AM

This AeroPress technique removes all the fuss from brewing, leaving you with a straightforward, reliable cup.

You'll need:
• Coffee: 11 g, ground on the finer side of medium.
• Water: 200 g at 94–96 °C.
• A scale, a timer, and a sturdy mug.

Method:
1. Insert a paper filter into the cap — no need to rinse or preheat.
2. Add the coffee to your AeroPress, set it on the mug, and place the whole setup on your scales.
3. Start the timer and pour all 200 g of water evenly over the grounds.
4. Insert the plunger about 1 cm to create a seal. Wait 2 minutes.
5. Gently swirl the brewer, then wait 30 seconds.
6. Press slowly over 20–30 seconds.

Clean-up is quick, and you're ready to enjoy your coffee.

Show less

♥ 2

@be9904　12/22/2025, 2:15:34 AM

No plans for winter vacation

♡ 0　Delete

Main Feed

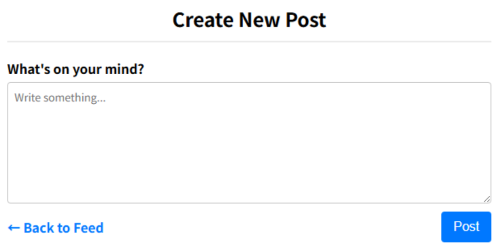
The file provides the main interface of the app, aggregating data from the backend into a readable format.

3. newpost.ejs

This file defines the create new post view. It serves as the user interface for authenticated users to draft and submit new content to the application.

The file contains three key structural components required for content creation.

- **Post Form** (HTML form): This form wraps the input elements and manages the data submission. It is configured with method="POST" and action="/createPost", directing the user's input to the server-side controller responsible for saving the new document to the database.
- **Content Input** (HTML textarea, required): This text input allows users to type their message. It is assigned the name="content" attribute, which corresponds to the variable extracted by the backend logic (req.body.content). The required attribute enforces client side validation, ensuring empty posts cannot be submitted.
- **Form Actions** (HTML div): This container organizes the interface controls. It includes the post submission button and a navigation link (Back to Feed), providing a clear exit path.


Create New Post

What's on your mind?

Write something...

← Back to Feed

Post

Create New Post

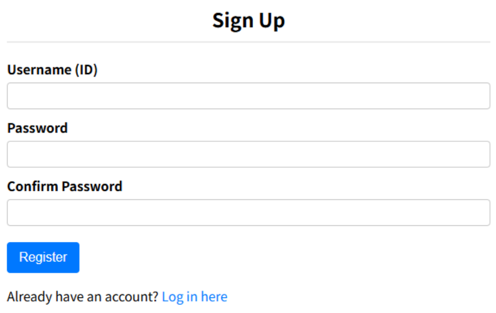
The file serves as the entry point for data generation.

4. signup.ejs

This file defines the signup view. It serves as the frontend interface for new user registration, allowing account creation and application access.

The file contains three key structural components required for the registration functionality.

- **Error Handling Block** (EJS condition): This section checks if an error variable was passed from the server (e.g., "Passwords do not match"). If present, it renders a warning message.
- **Registration Form** (HTML form): This form collects the new account details. It is configured with method="POST" and action="/signup", ensuring that the sensitive registration data is sent securely to the server's signup controller for processing.
- **Credential Inputs** (HTML inputs, Required): These fields capture the necessary user data: a unique id, a password, and a confirm password field. All are marked as required to prevent incomplete submissions, and the confirmation field allows the backend to verify that the user did not make a typo in their password. Additionally, more conditions were added to prevent creation of indistinguishable usernames such as 'skku' and 'skku ' or preserved names like 'admin'.


Sign Up

Username (ID)

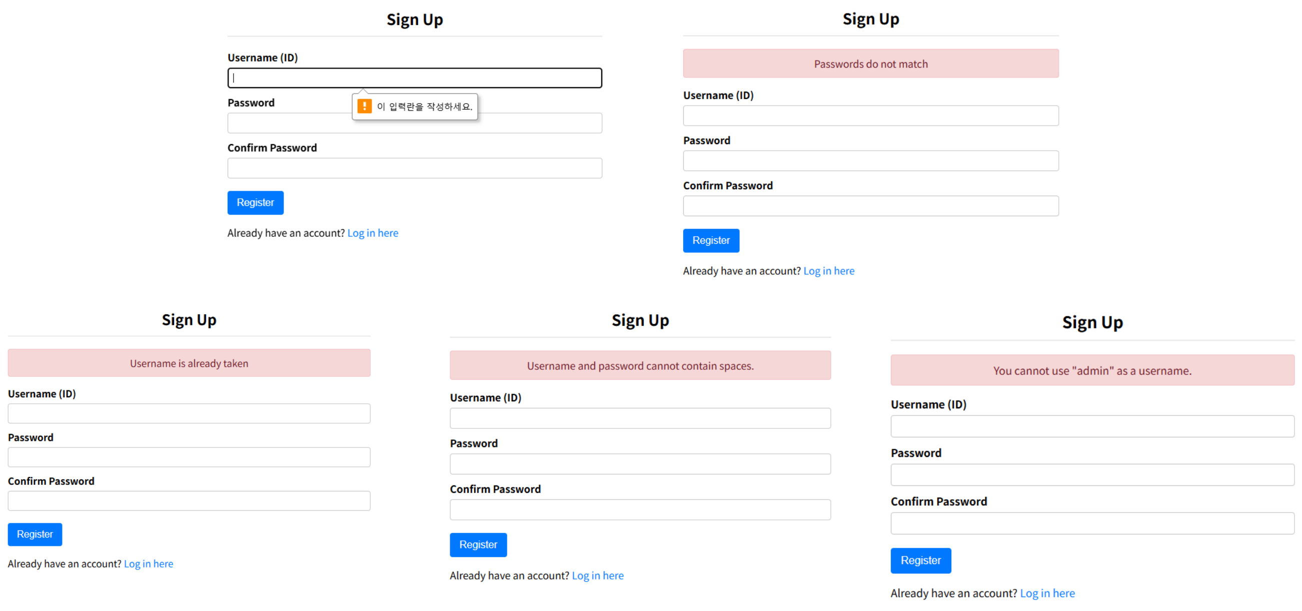
Password

Confirm Password

Register

Already have an account? Log in here

Signup Page


Sign Up

Username (ID)

Password

이 입력란을 작성하세요.

Confirm Password

Register

Already have an account? Log in here

Sign Up

Passwords do not match

Username (ID)

Password

Confirm Password

Register

Already have an account? Log in here

Sign Up

Username is already taken

Username (ID)

Password

Confirm Password

Register

Already have an account? Log in here

Sign Up

Username and password cannot contain spaces.

Username (ID)

Password

Confirm Password

Register

Already have an account? Log in here

Sign Up

You cannot use "admin" as a username.

Username (ID)

Password

Confirm Password

Register

Already have an account? Log in here

Signup Fail Cases

The file concludes with a navigation link redirecting existing users to the login page.

Other

1. /public/style.css

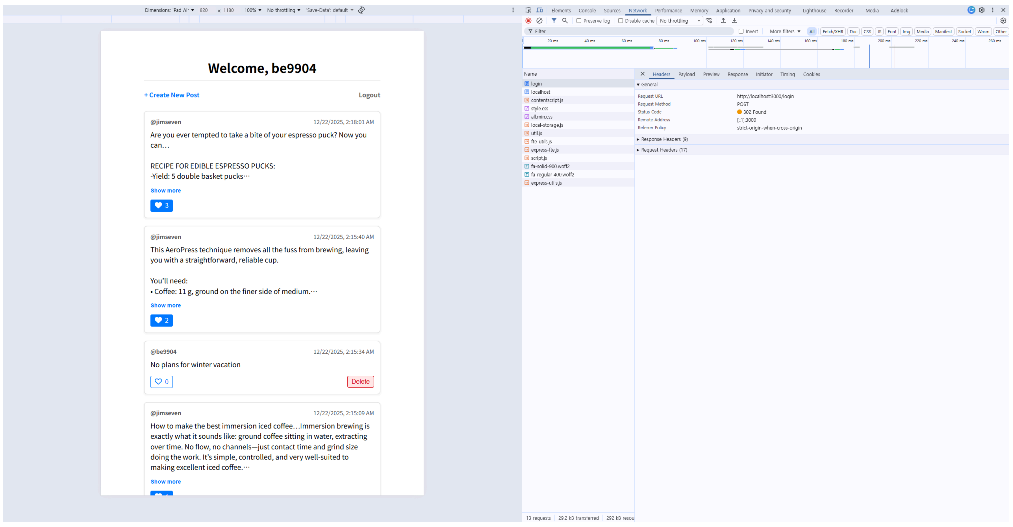
A stylesheet file used to define design layout of each page.

2. app.js

This file defines the application entry point. It serves as the central server file, bringing together database connections, middleware configuration, and routing logic for the application.

The file implements five key functional areas required for the backend architecture.

- **Dependencies and Setup** (Imports & Initialization): This section imports necessary modules such as express for the framework, mongoose for database interaction, bcrypt for password hashing, and express-session for state management. It also initializes the express application instance.
- **Middleware Configuration** (App configuration): This block configures the global middleware stack. It sets EJS as the view engine, enables body parsing for JSON and URL encoded data, serves static files from the public directory, and initializes session management using a secure secret key.
- **Database Connection** (Mongoose): This section establishes a connection to the local MongoDB instance (mongodb://127.0.0.1:27017/nodejs). It uses promises to log a success message upon connection or an error message if the connection fails.
- **Controller Logic** (Async functions): This core section defines the business logic for handling user requests:
    - *User Management*: `signupUser` and `loginUser` handle account creation (hashing passwords) and authentication (verifying hashes and setting session cookies). `logoutUser` destroys the active session.
    - *Post Management*: `createPost` saves new content to the database. `getPosts` retrieves and sorts posts for the feed. `likePost` handles toggle logic (incrementing/decrementing likes) via AJAX. `deletePost` enforces authorization checks before removing content.
- **Routing** (URL Mapping): This section maps HTTP endpoints (GET and POST) to their respective controller functions. It defines routes for rendering pages (login, signup, create new post) and handling actions (API calls), before finally starting the server on port 3000.


Dimensions: iPad Air ▾ 820 × 1180 100% ▾ No throttling ▾ 'Save-Data': default ▾
Elements Console Sources Network Performance Memory Application Privacy and security Lighthouse Recorder Media AdBlock
Preserve log Disable cache No throttling
Filter Invert More filters All Fetch/XHR Doc CSS JS Font Img Media Manifest Socket Wasm Other
Headers Payload Preview Response Initiator Timing Cookies
+ Create New Post
Logout
Show more
Delete

Welcome, be9904

@jimseven 12/22/2025, 2:18:01 AM
Are you ever tempted to take a bite of your espresso puck? Now you can…

RECIPE FOR EDIBLE ESPRESSO PUCKS:
-Yield: 5 double basket pucks···
3

@jimseven 12/22/2025, 2:15:40 AM
This AeroPress technique removes all the fuss from brewing, leaving you with a straightforward, reliable cup.

You'll need:
• Coffee: 11 g, ground on the finer side of medium.···
2

@be9904 12/22/2025, 2:15:34 AM
No plans for winter vacation
0

@jimseven 12/22/2025, 2:15:09 AM
How to make the best immersion iced coffee…Immersion brewing is exactly what it sounds like: ground coffee sitting in water, extracting over time. No flow, no channels—just contact time and grind size doing the work. It's simple, controlled, and very well-suited to making excellent iced coffee.···

```
Name
login
localhost
contentscript.js
style.css
all.min.css
local-storage.js
util.js
fte-utils.js
express-fte.js
script.js
fa-solid-900.woff2
fa-regular-400.woff2
express-utils.js

▾ General
Request URL       http://localhost:3000/login
Request Method    POST
Status Code       302 Found
Remote Address    [::1]:3000
Referrer Policy   strict-origin-when-cross-origin
▸ Response Headers (9)
▸ Request Headers (17)

13 requests | 29.2 kB transferred | 292 kB resou
```

Request Status after Login Success

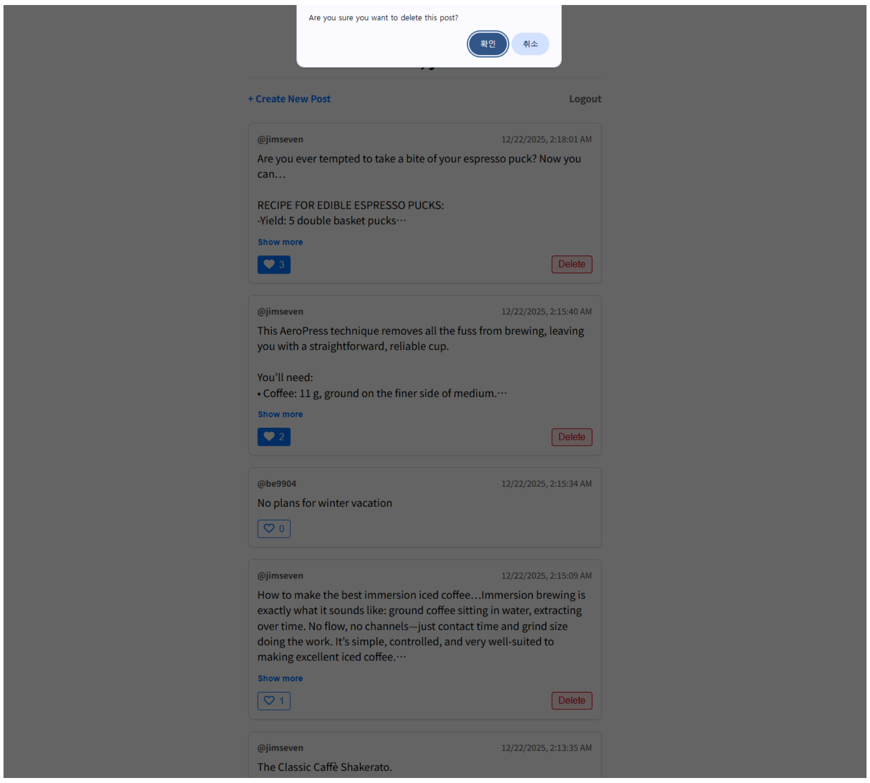
The file serves as the backend controller, tying together the database models, frontend views, and client-side requests into a cohesive functional application.

3. /public/script.js

This file defines the client side logic. It manages the interactive elements of the application that run directly in the user's browser, handling event listeners and asynchronous server communication.

The file implements three key features required for a responsive user interface.

- **Delete Confirmation** (Event listener): This block attaches a 'submit' listener to all forms marked with the class `.delete-form`. It intercepts the submission process to display a browser-native confirmation dialog (`confirm()`). If the user cancels the dialog, the script calls `e.preventDefault()` , aborting the request and preventing accidental data loss.



Are you sure you want to delete this post?

@jimseven 12/22/2025, 2:18:01 AM
Are you ever tempted to take a bite of your espresso puck? Now you can…

RECIPE FOR EDIBLE ESPRESSO PUCKS:
-Yield: 5 double basket pucks···
3

@jimseven 12/22/2025, 2:15:40 AM
This AeroPress technique removes all the fuss from brewing, leaving you with a straightforward, reliable cup.

You'll need:
• Coffee: 11 g, ground on the finer side of medium.···
2

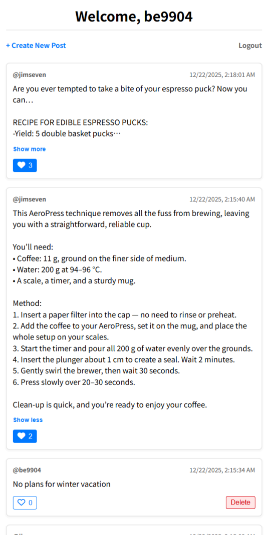
@be9904 12/22/2025, 2:15:34 AM
No plans for winter vacation
0

@jimseven 12/22/2025, 2:15:09 AM
How to make the best immersion iced coffee…Immersion brewing is exactly what it sounds like: ground coffee sitting in water, extracting over time. No flow, no channels—just contact time and grind size doing the work. It's simple, controlled, and very well-suited to making excellent iced coffee.···
1

@jimseven 12/22/2025, 2:13:35 AM
The Classic Caffè Shakerato.

Delete Confirmation on delete button click

- **Content Toggle** (Event Delegation): This section manages the show more/show less functionality for long posts. Instead of attaching listeners to every button individually, it uses a single listener on the document. When a click is detected on a `.btn-toggle-view` element, the script identifies the associated content container and toggles the `.collapsed` css class, updating the button text to match the current state.



Welcome, be9904

+ Create New Post
Logout

@jimseven 12/22/2025, 2:18:01 AM
Are you ever tempted to take a bite of your espresso puck? Now you can…

RECIPE FOR EDIBLE ESPRESSO PUCKS:
-Yield: 5 double basket pucks···
Show more
♥ 3

@jimseven 12/22/2025, 2:15:40 AM
This AeroPress technique removes all the fuss from brewing, leaving you with a straightforward, reliable cup.

You'll need:
• Coffee: 11 g, ground on the finer side of medium.
• Water: 200 g at 94–96 °C.
• A scale, a timer, and a sturdy mug.

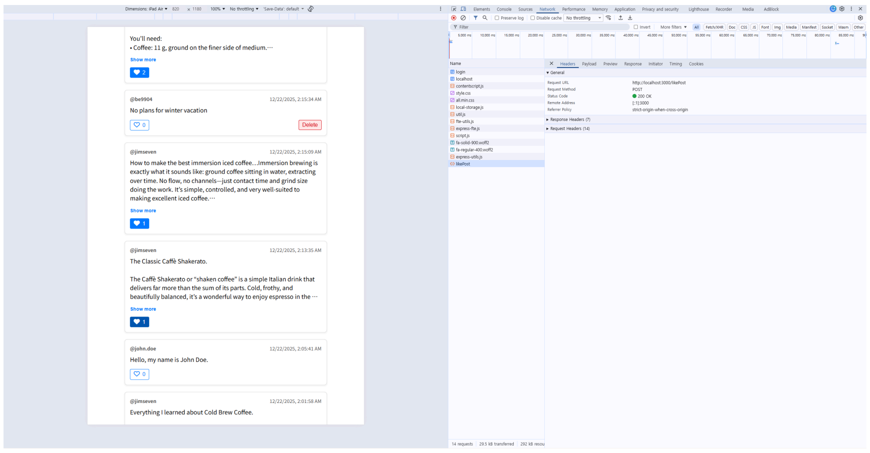
Method:
1. Insert a paper filter into the cap — no need to rinse or preheat.
2. Add the coffee to your AeroPress, set it on the mug, and place the whole setup on your scales.
3. Start the timer and pour all 200 g of water evenly over the grounds.
4. Insert the plunger about 1 cm to create a seal. Wait 2 minutes.
5. Gently swirl the brewer, then wait 30 seconds.
6. Press slowly over 20–30 seconds.

Clean-up is quick, and you're ready to enjoy your coffee.
Show less
♥ 2

@be9904 12/22/2025, 2:15:34 AM
No plans for winter vacation
♡ 0 Delete

Show More/Less for Long Posts

- **Asynchronous Like Toggle** (Async function): The asynchronous function `toggleLike` , enables users to like posts without reloading the page. It sends a POST request to /likePost using the fetch API. Upon receiving a json response from the server, it instantly updates the like count, toggling the button's `.liked` class, and swapping the font awesome icon between 'regular' (outline) and 'solid' (filled) to reflect the new state immediately.



Dimensions: iPad Air 820 × 1180 100% No throttling 'Save-Data': default

You'll need:
• Coffee: 11 g, ground on the finer side of medium.···
Show more
♥ 2

@be9904 12/22/2025, 2:15:34 AM
No plans for winter vacation
♡ 0 Delete

@jimseven 12/22/2025, 2:15:09 AM
How to make the best immersion iced coffee…Immersion brewing is exactly what it sounds like: ground coffee sitting in water, extracting over time. No flow, no channels—just contact time and grind size doing the work. It's simple, controlled, and very well-suited to making excellent iced coffee.···
Show more
♥ 1

@jimseven 12/22/2025, 2:13:35 AM
The Classic Caffè Shakerato.

The Caffè Shakerato or "shaken coffee" is a simple Italian drink that delivers far more than the sum of its parts. Cold, frothy, and beautifully balanced, it's a wonderful way to enjoy espresso in the ···
Show more
♥ 1

@john.doe 12/22/2025, 2:05:41 AM
Hello, my name is John Doe.
♡ 0

@jimseven 12/22/2025, 2:01:58 AM
Everything I learned about Cold Brew Coffee.

```
Name
login
localhost
contentscript.js
style.css
all.min.css
local-storage.js
util.js
fte-utils.js
express-fte.js
script.js
fa-solid-900.woff2
fa-regular-400.woff2
express-utils.js
likePost
```

Headers Payload Preview Response Initiator Timing Cookies

```
▼ General
Request URL         http://localhost:3000/likePost
Request Method      POST
Status Code         ● 200 OK
Remote Address      [::1]:3000
Referrer Policy     strict-origin-when-cross-origin
▶ Response Headers (7)
▶ Request Headers (14)
```

Async Like without Reloading Page

The file serves as the interactive layer, bridging the gap between static HTML views and dynamic user actions to allow smooth execution.

4. createAdmin.js

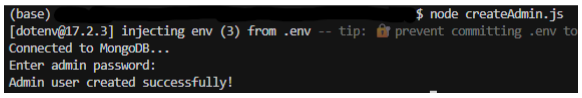
This file is the administrative seeding script. It is a utility designed to bypass standard registration restrictions and manually inject a privileged user account into the database. On the initial run of this app, this script should be run with

```
$node createAdmin.js
```

once to create an admin account with the admin role, since using the signup page blocks the creation of any account with id 'admin'.

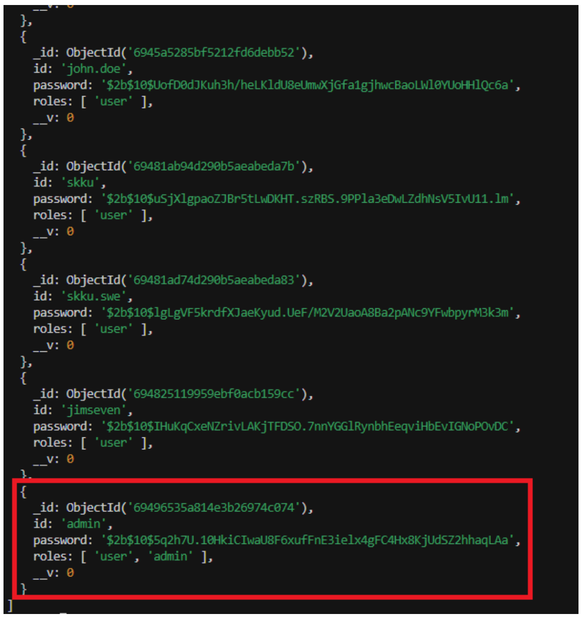
The script implements four key steps required to safely create the admin user:

- **Raw Mode Input Handling** (Custom function): This function manually manages the standard input stream (`process.stdin`) to create a secure password prompt. By setting `setRawMode(true)` , it intercepts individual keystrokes before they are rendered to the terminal, masking the input (similar to the Linux `sudo` behavior) while saving the password string in memory.
- **Password Security** (bcrypt implementation): Once the password is captured from the hidden prompt, the script uses `bcrypt` to hash it with a salt round of 10. This ensures that the manually created admin credentials satisfies to the application's security standards, preventing plain text storage.
- **Privileged Account Creation** (Object instantiation): The script constructs a user document with the hardcoded username `admin` . It explicitly grants the `['user', 'admin']` roles, allowing special permissions (like post deletion) that are inaccessible through the public signup form.
- **Connection Management** (Cleanup): This block handles the script's lifecycle using a `finally` clause. It guarantees that `mongoose.connection.close()` is called after the operation completes (whether successful or failed).



```
(base)                                                    $ node createAdmin.js
[dotenv@17.2.3] injecting env (3) from .env -- tip: 🔐 prevent committing .env to
Connected to MongoDB...
Enter admin password:
Admin user created successfully!
```

Adding admin account to DB via createAdmin.js



```
  },
  {
    _id: ObjectId('6945a5285bf5212fd6debb52'),
    id: 'john.doe',
    password: '$2b$10$UofD0dJKuh3h/heLKldU8eUmwXjGfa1gjhwcBaoLWl0YUoHHlQc6a',
    roles: [ 'user' ],
    __v: 0
  },
  {
    _id: ObjectId('69481ab94d290b5aeabeda7b'),
    id: 'skku',
    password: '$2b$10$uSjXlgpaoZJBr5tLwDKHT.szRBS.9PPla3eDwLZdhNsV5IvU11.lm',
    roles: [ 'user' ],
    __v: 0
  },
  {
    _id: ObjectId('69481ad74d290b5aeabeda83'),
    id: 'skku.swe',
    password: '$2b$10$lgLgVF5krdfXJaeKyud.UeF/M2V2UaoA8Ba2pANc9YFwbpyrM3k3m',
    roles: [ 'user' ],
    __v: 0
  },
  {
    _id: ObjectId('694825119959ebf0acb159cc'),
    id: 'jimseven',
    password: '$2b$10$IHuKqCxeNZrivLAKjTFDSO.7nnYGGlRynbhEeqviHbEvIGNoPOvDC',
    roles: [ 'user' ],
    __v: 0
  },
  {
    _id: ObjectId('69496535a814e3b26974c074'),
    id: 'admin',
    password: '$2b$10$5q2h7U.10HkiCIwaU8F6xufFnE3ielx4gFC4Hx8KjUdSZ2hhaqLAa',
    roles: [ 'user', 'admin' ],
    __v: 0
  }
]
```

Admin account successfully added to DB

The script executes immediately upon running, prompting the user for credentials only after a successful database connection is confirmed.