❖ Pacakages used:

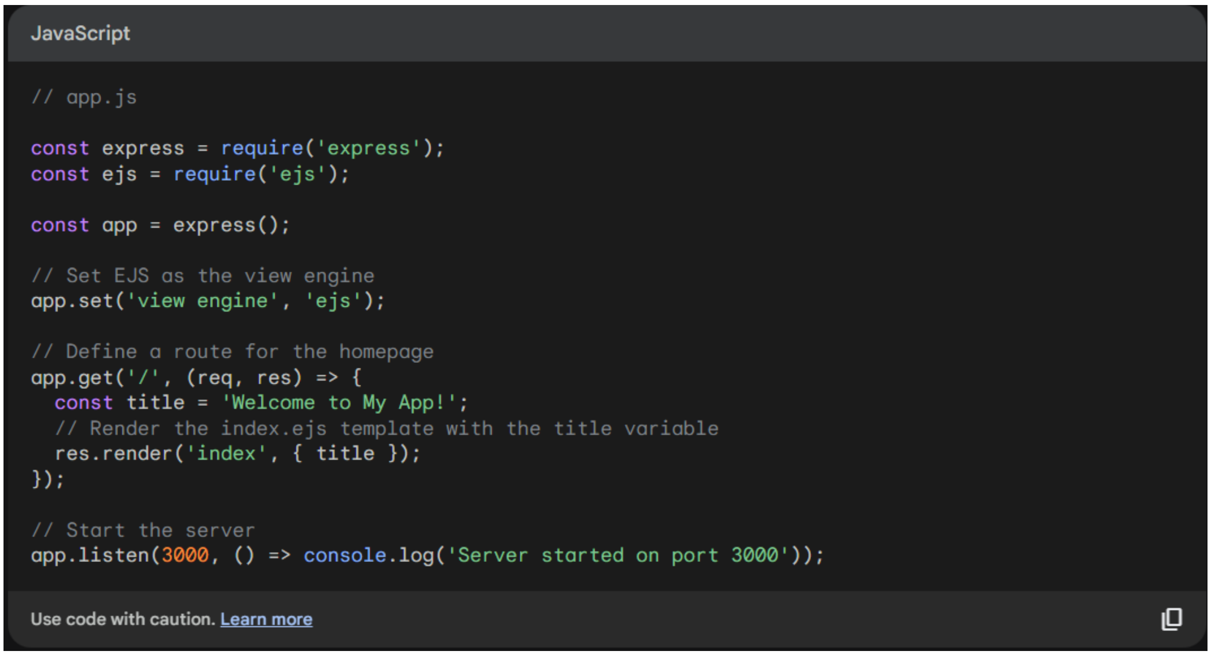
1. **Express**: The main Node.js framework for building web applications, handling routing, middleware, and more.
2. **EJS**: A templating engine that allows you to create dynamic HTML pages based on data.
3. **Mongoose**: An Object Data Modeling (ODM) library for interacting with MongoDB databases.

➢ Express.js Basics for your Presentation

- Node.js Framework: Express builds upon Node.js, making it simpler to handle HTTP requests and responses.
- Routing: Define the paths (URLs) your app responds to and the corresponding functionality.
- Middleware: Chain functions to pre-process requests or handle common tasks like logging or authentication.
- Templating: Integrate templating engines like EJS to dynamically generate HTML based on data.

Basic Template with Example:

Here's a simple Express example highlighting routing and templating:



JavaScript

```javascript
// app.js

const express = require('express');
const ejs = require('ejs');

const app = express();

// Set EJS as the view engine
app.set('view engine', 'ejs');

// Define a route for the homepage
app.get('/', (req, res) => {
  const title = 'Welcome to My App!';
  // Render the index.ejs template with the title variable
  res.render('index', { title });
});

// Start the server
app.listen(3000, () => console.log('Server started on port 3000'));
```

Use code with caution. Learn more

This code defines a route for the / path and renders the index.ejs template with a "Welcome" title. EJS allows injecting dynamic data (like the title) into the template for flexibility.

➢ **EJS View Engine Template in Express: A Brief Overview:**

What is it?
EJS, or Embedded JavaScript templates, is a popular templating engine used in conjunction with Express.js, a web development framework for Node.js. It allows you to write HTML with placeholders for dynamic content that gets filled later by JavaScript code.

Why use it instead of HTML pages?
There are several benefits to using EJS over plain HTML pages for your Express application:

- Dynamic Content: You can easily generate dynamic content based on data (e.g., user names, product lists, database results) within your templates.
- Code Reorganization: Complex logic can be moved out of HTML into separate JavaScript files, keeping your templates cleaner and easier to maintain.
- Conditional Rendering: Show or hide specific sections of your templates based on conditions, offering flexible layouts.
- Loops and Iterations: Loop through arrays of data and efficiently generate repetitive HTML elements.

Other Types of Template Engines:
There are numerous templating engines besides EJS, each with its own strengths and weaknesses. Here are a few popular options:

- Pug (formerly Jade)
- Handlebars
- Mustache
- Liqui

Why EJS is Popular for Beginners:
EJS is a particularly good choice for beginners due to several factors:

- Easy to Learn: EJS syntax is straightforward and similar to writing HTML, making it accessible for newcomers.
- Minimal Setup: It integrates seamlessly with Express, requiring minimal configuration for basic tasks.

❖ **MongoDB concepts:**

MongoDB Explained: Your NoSQL Database Playground

MongoDB is a popular NoSQL (Not Only SQL) database management system. Unlike traditional relational databases, it stores data in flexible, document-like structures instead of rigid tables and rows. This makes it a powerful tool for:

- Highly Scalable Applications: Easily handle large volumes of data and adapt to growing user bases.
- Dynamic Data Storage: Store unstructured and diverse data efficiently, including text, images, and even nested objects.
- Rich Querying: Perform complex queries using familiar JavaScript syntax, even on nested data structures.
- High Performance: Achieve fast read and write operations thanks to its document-oriented structure and indexing capabilities.

➢ MongoDB Concepts(Collection,Schema,Document,Model) In easy words:

Imagine you're building a library to store books. Let's break down the terms you mentioned:

- **Collection**: Think of it as a big room in your library. It holds all the books on a specific topic, like fiction or biographies. In MongoDB, a collection is a group of documents that share a common theme.
- **Document**: This is the actual book itself! It contains all the information about a single book, like title, author, and publication date. In MongoDB, a document is a set of key-value pairs, like a JSON object.
- **Schema**: This is like the blueprint for your books. It defines what information each book should have, like title, author, and pages. In MongoDB, a schema is an optional blueprint that defines the structure of documents in a collection. It's not strictly required, but it helps ensure consistency and makes your data easier to manage.
- **Model**: This is like a pre-built book template. It takes the schema and makes it easier to create new documents. In MongoDB, a model is a JavaScript object that represents a document in a collection. You can use models to create, read, update, and delete documents easily.

Here's an analogy:

- Collection: Library room (e.g., fiction, biographies)

- Document: Book (e.g., "The Lord of the Rings", "Steve Jobs")
- Schema: Book template (defines title, author, pages)
- Model: Pre-built book form (easy to create new books)

Project working

➢ Working of routes/index.js file:

index.js Breakdown without Analogies:

This file defines routes for your web application using Express.js:

1. Dependencies:
   - express: Creates the foundation for routing requests and responses.
   - router: An Express object used to manage specific routes.
   - data.js: Contains data for recommendations (e.g., music, movies, books).
   - userModel: Mongoose model representing users' data (username, password, etc.).
   - notesModel: Mongoose model representing users' notes (title, content, etc.).
2. Route Handlers:
   - GET /getdata/:user: Retrieves users' mood data or returns an error if user is not found.
   - GET /, POST /login: Login functionality with user authentication and redirects based on success/failure.
   - POST /register: Creates a new user and redirects to the home page.
   - GET /register, /login, /logout: Render corresponding EJS templates for registration, login, and logout pages.
   - GET /mood_update/:Currentmood/:user: Updates a user's current mood in their data.
   - GET /home/:user, GET /journal/:user: Serves home and journal pages by rendering EJS templates with user data.
   - GET /home/chatbot: Renders the chatbot EJS template.
   - GET /music/:user, /movie/:user, /book/:user: Fetches recommendations based on user's mood and renders the recommend EJS template.
   - GET /profile/:user: Retrieves user information and renders the profile EJS template.
   - POST /add-entry/:user: Creates a new note for a user and redirects to the journal page.
3. Helper Functions:
   - `getRandomIndices, getRandomItem:** These functions select random items from the "data.js" based on mood and category for recommendations.
4. Exports:
   - module.exports = router: Makes the router object available for use in other modules of your application.

❖ Simple realtime example for index.js file:

Here's a breakdown of your index.js file in a noob-friendly way:
Imagine a train station with different routes leading to various destinations. This file is like the station master, deciding which trains go where.
Here's how it works:

1. Getting Ready:
   - It grabs a train engine named express to get things moving.

    - It creates a blueprint for train routes called router.
    - It picks up some helpful tools for managing passengers (data) and luggage (notes).
2. Main Routes:
    - Home (/): Welcomes passengers and checks their tickets (user IDs). If they have a valid ticket, they're directed to the main platform (index.ejs). If not, they're sent to the ticket office (login).
    - Registration (/register): Helps new passengers create accounts and get their tickets.
    - Login (/login): Verifies passengers' tickets and lets them board the train.
    - Mood Updates (/mood_update/:Currentmood/:user): Allows passengers to choose their mood for the journey, like picking a seat.
    - Home Platform (/home): The central hub where passengers can explore different areas of the station.
    - Recommendations (/music, /movie, /book): Suggests entertainment options based on passengers' moods, like offering snacks and drinks on the train.
    - Profile (/profile): Displays a passenger's personal information, like checking their passport.
    - Journal (/journal): Lets passengers write down their thoughts and feelings during the journey, like a travel diary.
    - Adding Journal Entries (/add-entry): Helps passengers create new entries in their journals.
    - Chatbot (/home/chatbot): Offers a friendly chatbot to chat with passengers, like a helpful conductor.

3. Behind the Scenes:
    - Finding Data (getRandomItem): This function digs through the station's storage to find specific items, like fetching snacks from different compartments.
    - Packing Luggage (getRandomIndices): This function helps organize passenger luggage efficiently, like stacking suitcases neatly.
4. Departing the Station:
    - The module.exports = router; line sends the train routes out into the world, ready to guide passengers to their destinations!

Remember:

- User IDs: These are like special tickets that passengers need to access different parts of the station.
- MongoDB Models: These are like maps of the train's compartments, helping the station master find things quickly.
- Express Rendering: This is like announcing train departures and helping passengers find their seats on the train.

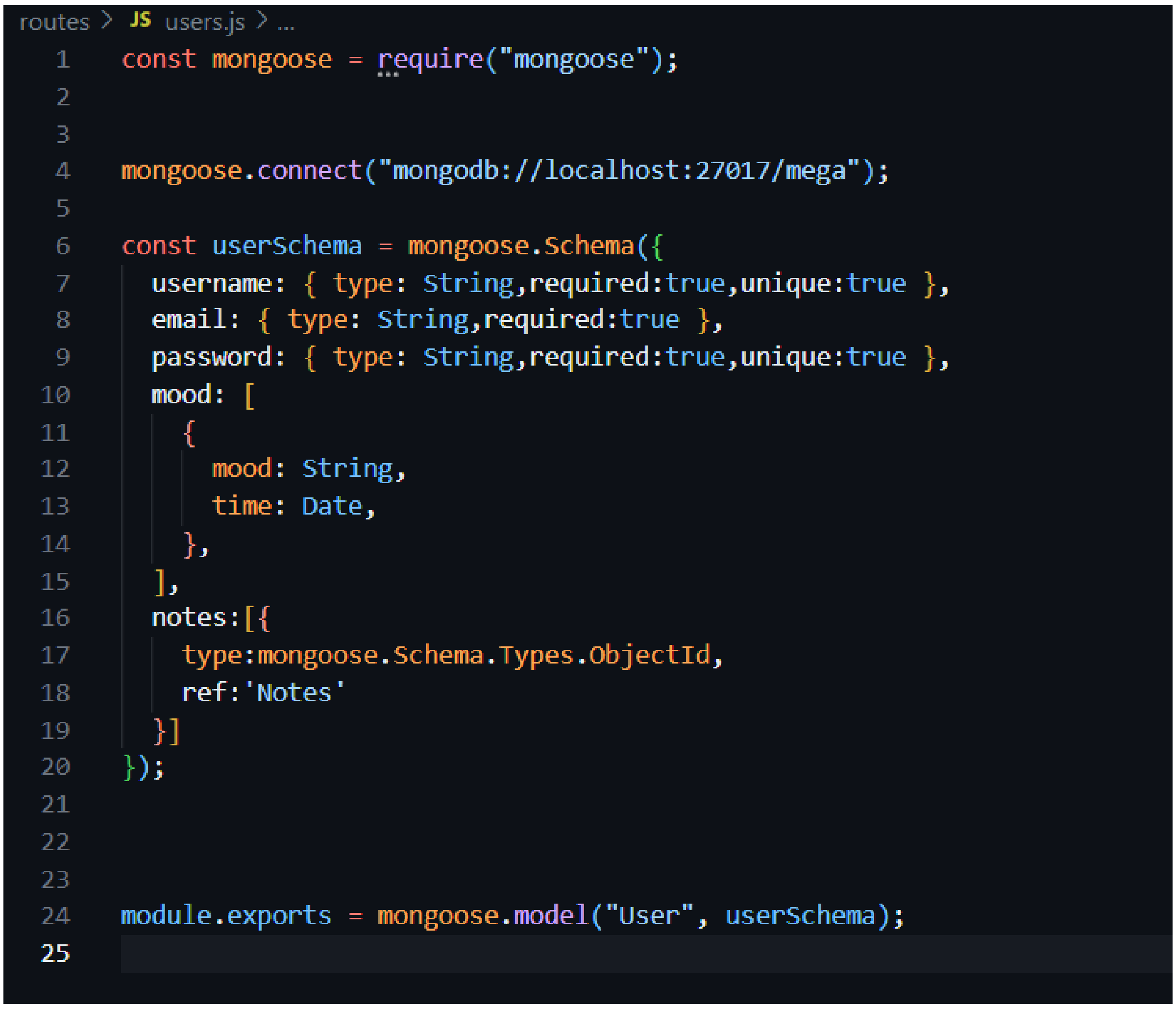
➢ **Data.js:**

This file, data.js, stores a large collection of recommendations for different categories like music, books, and movies, categorized by mood. It's like a big library of entertainment options tailored to your emotions!

Here's a breakdown:

- Structure: It's a JavaScript object with three main categories: music, book, and movie.
- Subcategories: Each category has subcategories like happy, sad, angry, joyful, and depress.
- Content: Each subcategory contains a list of titles in that category and mood. For example, under music, happy has entries like "Happy" by Pharrell Williams and "Walking On Sunshine" by Katrina & The Waves.

➢ **routes/user.js file:**


routes > JS users.js > ...

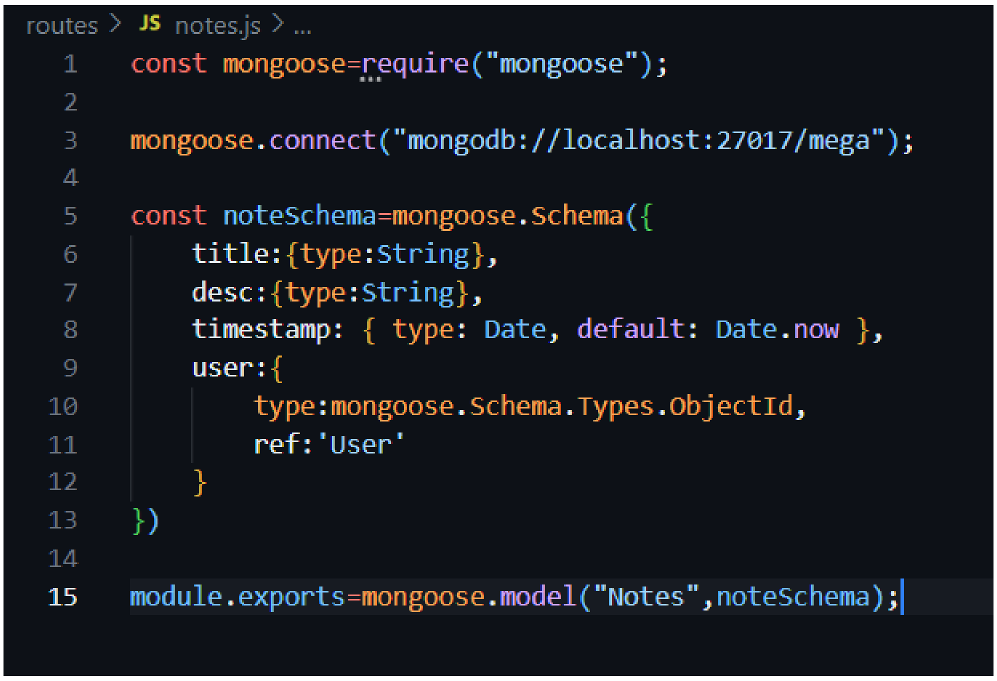
```
const mongoose = require("mongoose");


mongoose.connect("mongodb://localhost:27017/mega");

const userSchema = mongoose.Schema({
  username: { type: String,required:true,unique:true },
  email: { type: String,required:true },
  password: { type: String,required:true,unique:true },
  mood: [
    {
      mood: String,
      time: Date,
    },
  ],
  notes:[{
    type:mongoose.Schema.Types.ObjectId,
    ref:'Notes'
  }]
});



module.exports = mongoose.model("User", userSchema);

```

Here's a simple explanation of the models:

❖ **user.js:**

- Purpose: Defines a blueprint for storing user information in the database.
- Steps:
  1. Imports Mongoose: Connects to MongoDB, a database for storing data.
  2. Connects to MongoDB: Establishes a connection to a database named "mega".
  3. Creates User Schema: Defines the structure for user data:
     - username: Unique text for identifying users.
     - email: User's email address.
     - password: User's password (stored securely).
     - mood: List of objects containing mood strings and timestamps, tracking mood changes.
     - notes: List of references to notes created by the user.
  4. Exports User Model: Makes the model available for use in other parts of the application.

❖ notes.js:



routes > JS notes.js > ...

```javascript
const mongoose=require("mongoose");

mongoose.connect("mongodb://localhost:27017/mega");

const noteSchema=mongoose.Schema({
    title:{type:String},
    desc:{type:String},
    timestamp: { type: Date, default: Date.now },
    user:{
        type:mongoose.Schema.Types.ObjectId,
        ref:'User'
    }
})

module.exports=mongoose.model("Notes",noteSchema);
```

1. Purpose: Defines a blueprint for storing individual notes in the database.
2. Steps:
    1. Imports Mongoose: Connects to MongoDB.
    2. Connects to MongoDB: Same connection as in user.js.
    3. Creates Note Schema: Defines the structure for notes:
        - title: Textual title of the note.
        - desc: Main content of the note.
        - timestamp: Date and time when the note was created, automatically set.
        - user: Reference to the user who created the note, linking notes to users.
    4. Exports Notes Model: Makes the model available for use.

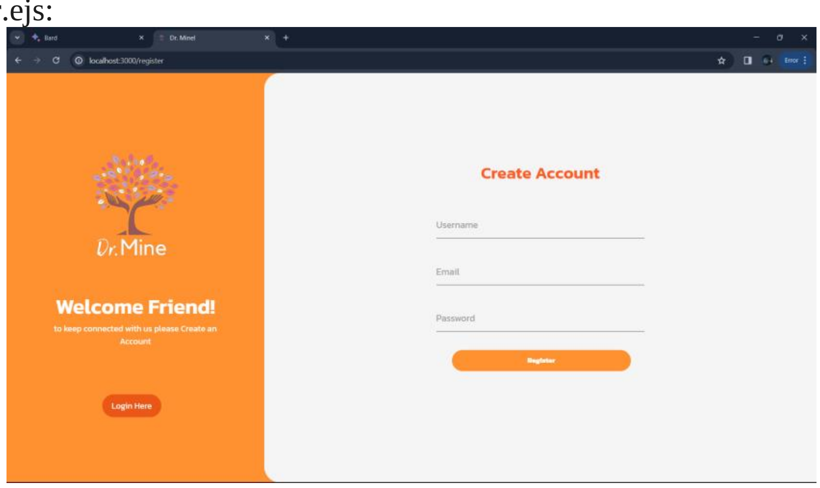
Key takeaways:

- Models act as templates: They define how data is organized and stored in the database.
- Schemas specify structure: They define the types of data and relationships within a model.
- Mongoose connects to MongoDB: It handles interactions with the database.

- Models are reusable: They can be used throughout the application to create, read, update, and delete data.

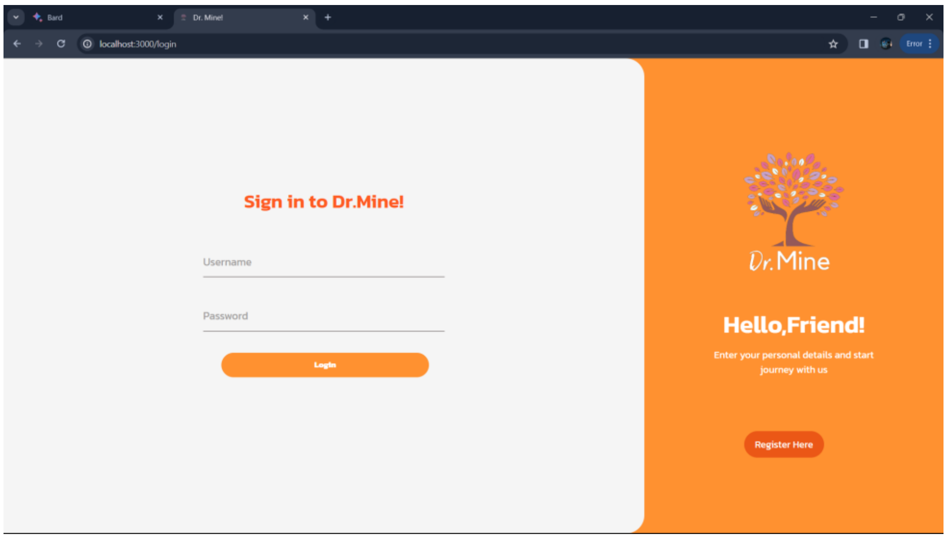
Views Folder Working

1. register.ejs:



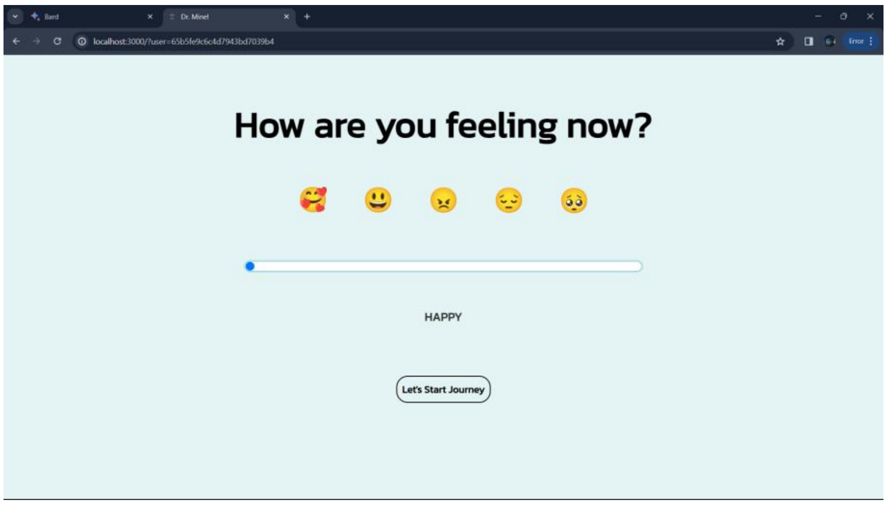
- Purpose: Creates a visually appealing registration page for new users.
- Key Elements:
  - A form for entering username, email, and password.
  - A logo and welcoming text.
  - A link for users who already have accounts to log in.
  -

3. login.ejs:
4.



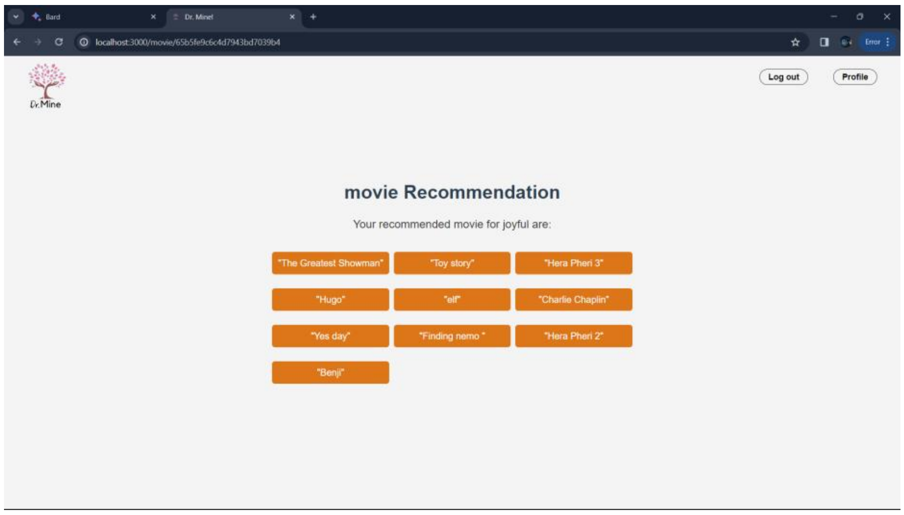
- Purpose: Handles user logins with a user-friendly interface.
- Key Elements:
  - A form for entering username and password.
  - A logo and informative text.
  - A link for those who don't have accounts to register.
  -

3. index.ejs:



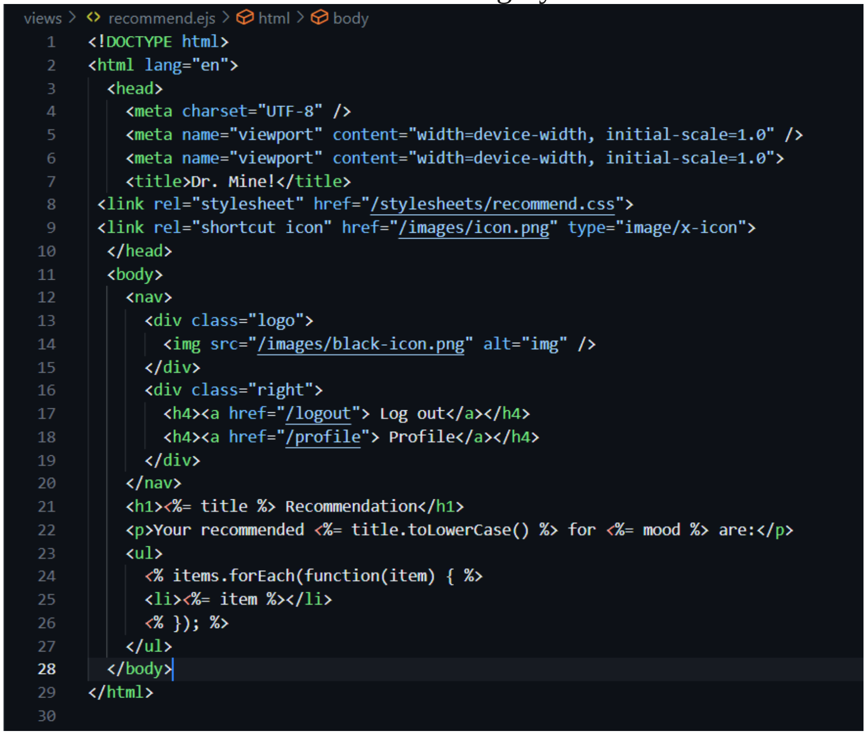
- Purpose: Captures the user's current mood to tailor recommendations.
- Key Elements:
  - A question asking "How are you feeling now?"
  - A collection of emojis representing different moods, each linked to a mood update route.
  - A slider for fine-tuning mood selection.
  - A button to initiate the recommendation journey.

4.recommend.ejs (for movie, music, and books):



- Purpose: Displays personalized recommendations based on the user's mood.
- Key Elements:
  - A navigation bar with a logo, logout link, and profile link.

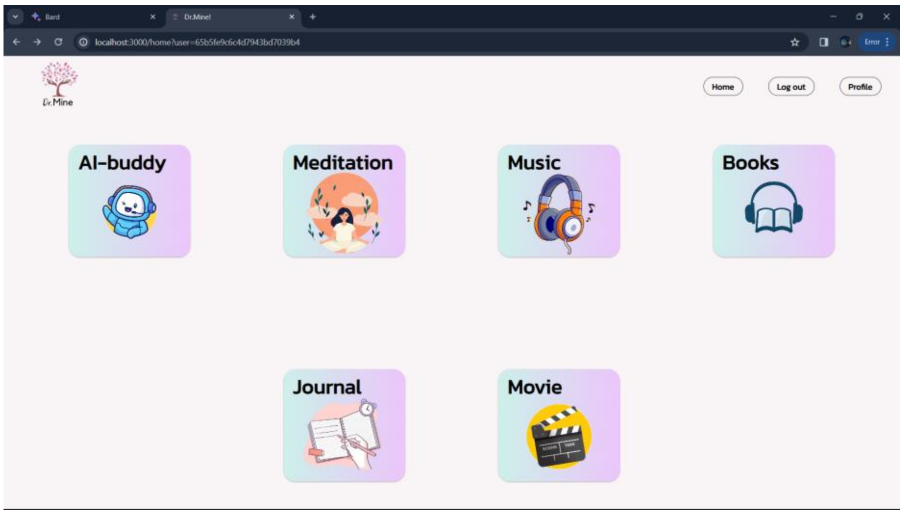
- A title indicating the category of recommendations (e.g., "Movie Recommendation").
- A sentence stating the user's mood and the recommendations being provided. A list of recommended items in that category

5. 

views > recommend.ejs > html > body

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dr. Mine!</title>
 <link rel="stylesheet" href="/stylesheets/recommend.css">
 <link rel="shortcut icon" href="/images/icon.png" type="image/x-icon">
  </head>
  <body>
    <nav>
      <div class="logo">
        <img src="/images/black-icon.png" alt="img" />
      </div>
      <div class="right">
        <h4><a href="/logout"> Log out</a></h4>
        <h4><a href="/profile"> Profile</a></h4>
      </div>
    </nav>
    <h1><%= title %> Recommendation</h1>
    <p>Your recommended <%= title.toLowerCase() %> for <%= mood %> are:</p>
    <ul>
      <% items.forEach(function(item) { %>
      <li><%= item %></li>
      <% }); %>
    </ul>
  </body>
</html>

```

5.home.ejs:



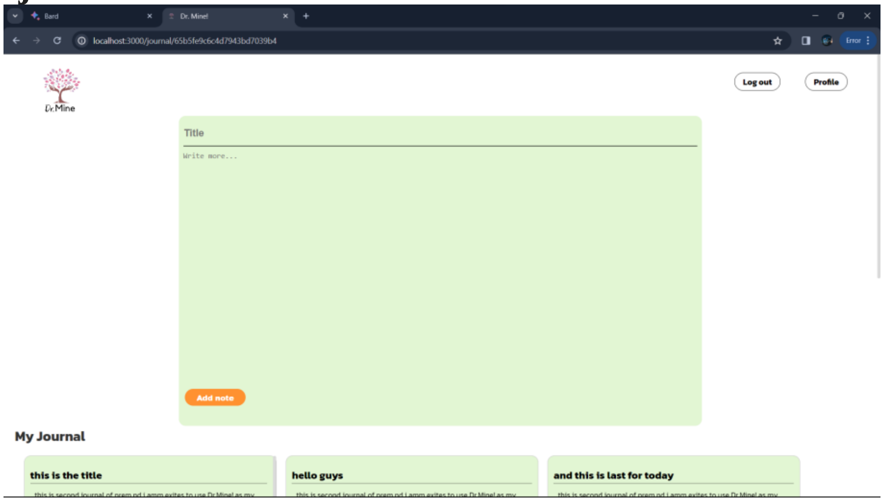
I'll break down the home.ejs file for you:

1. Structure and Key Elements:

- Basic HTML Structure: The file establishes the core structure of an HTML webpage using <!DOCTYPE html>, <html>, <head>, and <body> tags.

- Loader and Quote: It includes a temporary loading screen with an encouraging quote to engage users while the page loads.
- Navigation Bar: It features a navigation bar with:
    - The application logo.
    - Links for Home, Logout, and Profile (dynamically linked to the user's profile).
- Grid of Feature Boxes: It visually presents six core features of the application:
    - AI-buddy
    - Meditation
    - Music
    - Books
    - Journal
    - Movie
- External JavaScript: It incorporates two JavaScript files for functionality:
    - gsap.min.js: A popular animation library likely used to create smooth interactions and visual effects.
    - home.js: Custom JavaScript code specific to this page's functionality.
- The home.js file likely contains JavaScript code that:
    - Manages the loading screen and quote display.
    - Implements any interactive elements or animations on the page.
    - Handles feature box interactions and navigation.

7.Journal.ejs:



Here's a breakdown of the journal.ejs file:

Structure and Key Elements:

- Journal Entry Form: - Includes a title input field. - Provides a large textarea for content. - Offers a submit button to create new journal entries.
- Journal Display: - Lists existing journal entries with titles, content, and timestamps.

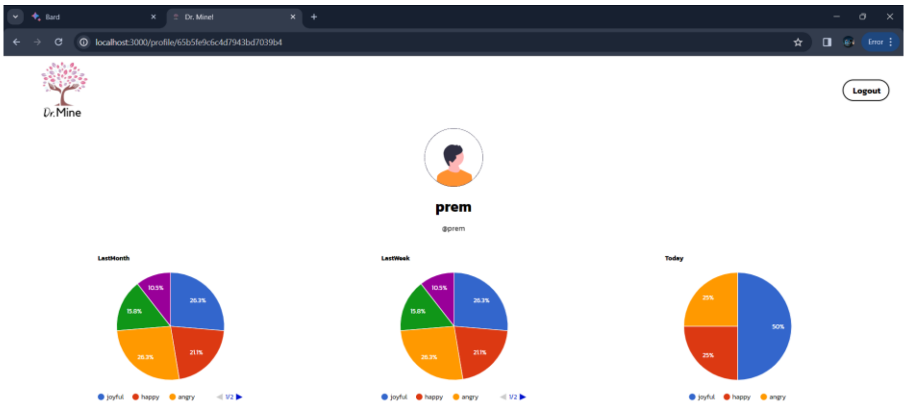
EJS Placeholders:

- <%= userID %>: Injects the current user's ID into the form action and likely in other parts of the page for personalization.
- <% user.notes.forEach(note => { %> ... <% }); %>: Dynamically iterates through the user's journal entries provided by the server-side logic.

Functionality:

- Journal Entry Creation: Upon form submission, the data (title and content) is sent to the /add-entry/<span class="math-inline">\{userID\}\ route for processing and storing the new journal entry. - **Journal Display:** Fetches and displays the user's existing journal entries, likely from a database or other data source. **Visual Design:** - Links to a separate stylesheet `journal.css` for visual styling. - Likely features a clean and organized layout, making it easy to read and write journal entries. **Key Points:** - The `journal.ejs` file handles both the creation and display of journal entries. - It relies on server-side routes (`/add-entry/{userID}` and likely others) to manage data storage and retrieval.

6.profile.ejs and profile.js file:



Here's a breakdown of the profile.ejs and profile.js files:

➢ **profile.ejs:**

Structure and Key Elements:

- User Details: It prominently displays the user's profile picture, name, and username.
- Chart Containers: It includes three designated areas (with IDs "LastMonth", "LastWeek", "Today") where the pie charts will be dynamically rendered.

- Google Charts Integration: It loads the Google Charts library for enabling visualizations.
- External JavaScript: It links to profile.js for the core functionality of fetching data and drawing charts.

EJS Placeholder:

- <%= username %>: This placeholder injects the current user's username into the page, personalizing the experience.

➢ **profile.js:**

Functionality:

- Retrieving User ID: It extracts the user ID from the URL path to fetch personalized data.
- Loading Google Charts Library: It ensures the necessary Google Charts API is loaded for chart creation.
- Fetching User Data: It makes an asynchronous request to /getdata/<span class="math-inline">\{userID\}\ to retrieve the user's mood history. - **Processing Mood Data:** - It sorts the received moods based on timestamps. - It filters moods for the last month, last week, and today. - It counts the occurrences of each mood within these periods. - **Drawing Pie Charts:** - It calls the `drawChart` function to create pie charts for each time period, using the filtered mood counts and specified chart options. - **Helper Functions:** - `countMoods`: Counts the occurrences of each mood in a given array. - `drawChart`: Creates a pie chart with the provided data and options. **Key Points:** - The `profile.ejs` file sets up the visual layout and placeholders for the profile page. - The `profile.js` file handles the dynamic aspects, fetching user data, and rendering the pie charts using Google Charts. - The backend route `/getdata/{userID}` is responsible for providing the user's mood data in the expected format.