DATA-236 Sec 21 & 71 - Distributed Systems for Data Engineering HOMEWORK 1

Vimalanandhan Sivanandham 017596436

GitHub link for full code artifacts - <https://github.com/Vimalanandhan/DATA-236---Distributed-Systems-for-Data-Engineering>

Instructions:

* Please provide the code solution for each question along with its intended output.

Ensure that the code and corresponding output screenshots are placed together, one below the other.

* Screenshots must be provided for the output of each question.
* Submission should be in PDF Format Questions:

1. HTML
   1. Create an HTML page with the title "HW1-{Your Name}"

A black screen with white text

AI-generated content may be incorrect.

A black screen with text

AI-generated content may be incorrect.

* 1. Add a heading tag to name the title of the blog topic. (Note. you have to use the header tag which renders the biggest font size)

A screenshot of a computer

AI-generated content may be incorrect.



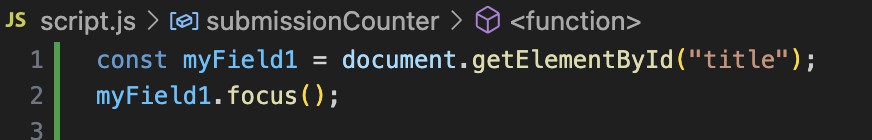
* 1. Create a form for the blog. The form should include the following:
     1. A text input for the blog title, placeholder text “Enter the title of your blog”. This field should be required and the cursor should automatically focus on this field when the page loads.

A screenshot of a computer

AI-generated content may be incorrect.

Required field:

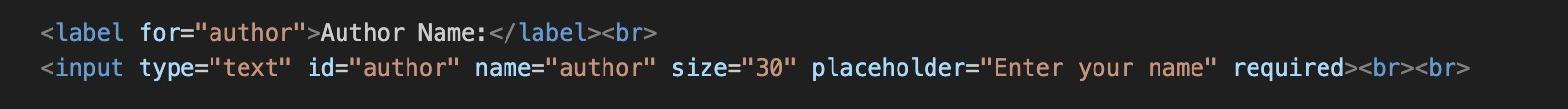
A screen shot of a computer

AI-generated content may be incorrect.

* + 1. A text input for the author name, placeholder "Enter your name”, and required.

A screenshot of a computer

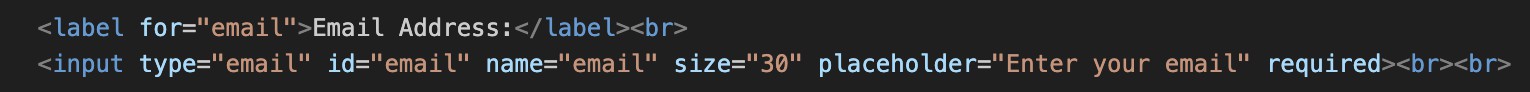
AI-generated content may be incorrect.



* + 1. An email input for the email address, placeholder “Enter your email”, and required.

A screenshot of a computer

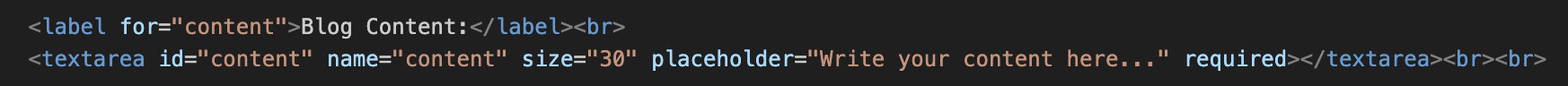
AI-generated content may be incorrect.



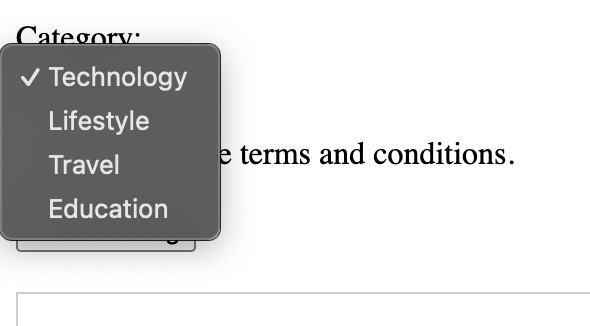
* + 1. A text area for the blog content, placeholder “Write your content here...”, and required.

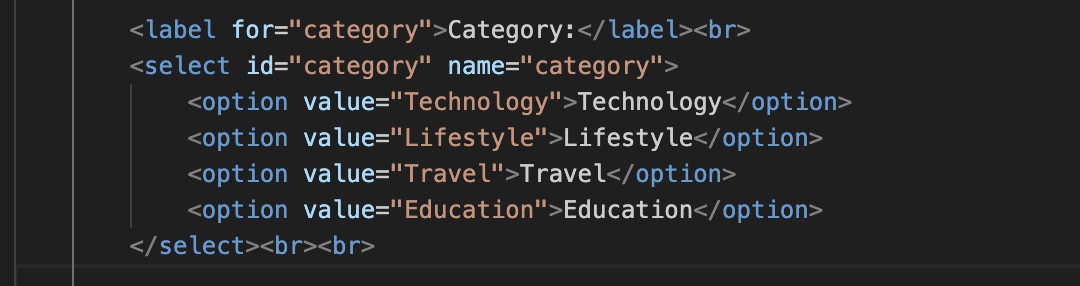
A screenshot of a computer

AI-generated content may be incorrect.



* + 1. A dropdown for category selection, and options "Technology," "Lifestyle," "Travel," and "Education."

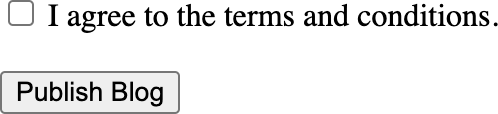




* + 1. A checkbox and a label with the text “I agree to the terms and conditions.”



* + 1. A submit button with the text “Publish Blog”.



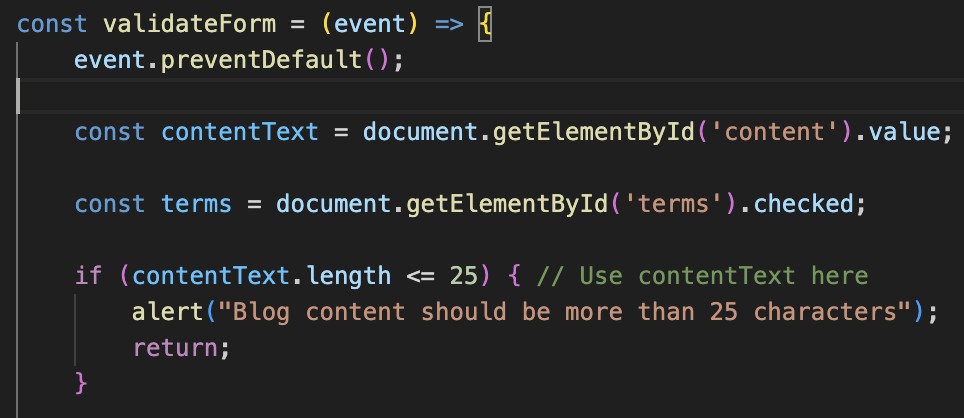
* 1. Add a script tag to link your javascript code for part II at the end of your HTML file.



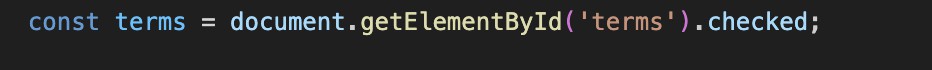
1. Javascript
   1. Write a javascript function using an arrow function to validate:
      1. Verify if the blog content is more than 25 characters. If the validation fails, display an alert with the message “Blog content should be more than 25 characters”.

A screenshot of a computer

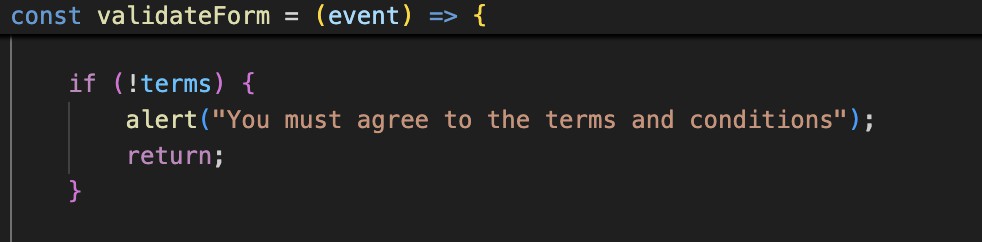
AI-generated content may be incorrect.



* + 1. Verify if the terms and conditions checkbox is checked. If the validation fails, display an alert with the message “You must agree to the terms and conditions”.

A screenshot of a computer

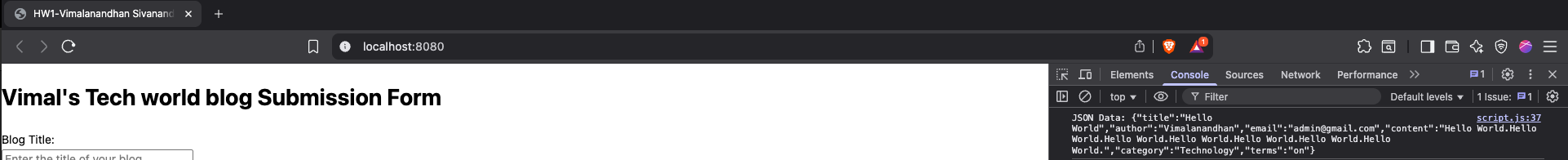
AI-generated content may be incorrect.

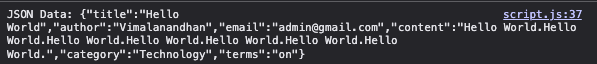


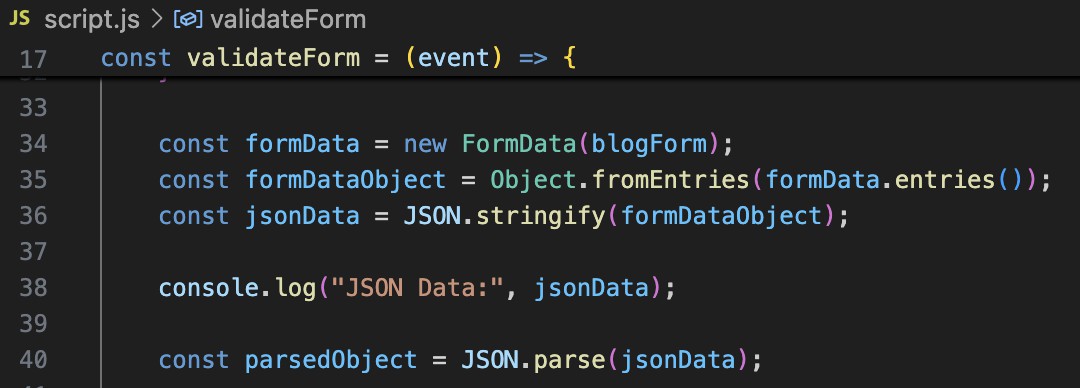
* 1. After the form submission is successful, convert the form data into a JSON string and log the output in the console.

A screenshot of a computer

AI-generated content may be incorrect.







* 1. Use object destructuring to extract the title and email fields from the parsed object and log their values in the console

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

* 1. Use the spread operator to add a new field “submissionDate” with the current date and time to the parsed object. Log the updated parsed object in the console.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

* 1. Create a closure to track how many times the form has been successfully submitted and log the submission count each time the form is submitted.

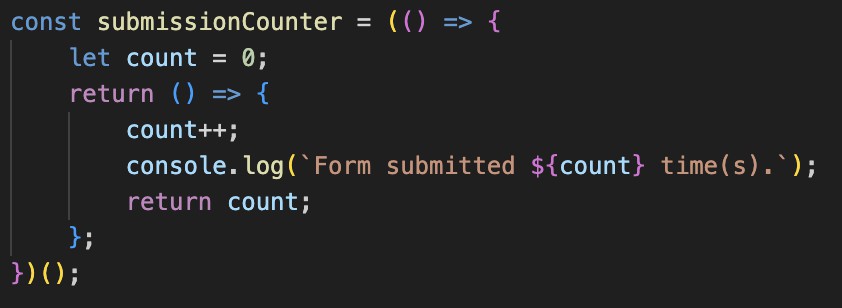
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.





# DEPLOYMENT:

Docker: Create a docker image of the above application and build and run the application using docker in your local.

A screenshot of a computer program

AI-generated content may be incorrect.



Provide the screenshot of your app running on your localhost.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# AWS ECS:

Create an AWS ECS service (only one task) running the above application using the docker image created above.

1. Create an ECR Repository. Go to the AWS Management Console > Elastic Container Registry (ECR). Click Create Repository

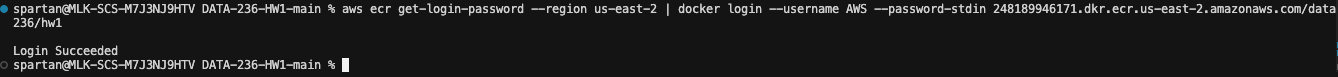
A screenshot of a computer

AI-generated content may be incorrect.

1. Authenticate Docker to ECR

aws ecr get-login-password --region <your-region> | docker login --username AWS

--password-stdin <your-account-id>.dkr.ecr.<your-region>.amazonaws.com



1. Tag and push the docker image docker tag <repo\_name>:latest

<your-account-id>.dkr.ecr.<your-region>.amazonaws.com/<repo\_name>:latest



docker push <your-account-id>.dkr.ecr.<your-region>.amazonaws.com/<repo\_name>:latest

A computer screen with white text

AI-generated content may be incorrect.

# Deploy to AWS ECS

1. Create an ECS Cluster:

Go to Elastic Container Service (ECS) > Create Cluster.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

1. Create a Task Definition:

Go to Task Definitions > Create new Task Definitions. Choose FARGATE > Name it.

Add a container:

Name: <container\_name>

Image URI: Use the ECR image URI (e.g.,

<your-account-id>.dkr.ecr.<your-region>.amazonaws.com/<repo\_name>:latest). Port mappings: Add port 80.

Click Create.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Create a Service Create an ECS Service:

Go to your cluster > Create Service.

Choose FARGATE > Select the task definition. Configure:

Service name: <service\_name>

Number of tasks: 1 Configure networking:

VPC: Choose your VPC.

Subnets: Choose at least two subnets.

Security group: Allow inbound traffic on port 80. Click Create Service.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Access Your Application:

Go to the Tasks tab in your ECS cluster. Find the public IP address of the task. Visit http://<public-ip> in your browser.

Provide the screenshot of your app running on your public IP address.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Clean Up to avoid unnecessary costs

1. Delete the ECS Service:

A screenshot of a computer

AI-generated content may be incorrect.

1. Delete the ECR Repository:

A screenshot of a computer

AI-generated content may be incorrect.

## **Agentic AI (Part 2)**

## Build two tiny “agents” that talk to each other using a small local LLM (via Ollama).

## Input: a blog title and content. Output: exactly 3 topical tags + a ≤25-word summary — produced through a short Planner → Reviewer → Finalizer flow — and printed as valid JSON.

## Requirements: ● Python 3.11+ (3.11 or 3.12 recommended). Note: 3.13 has compatibility issues with langchain and numpy(internal dependency) ● Ollama installed and running. ● A local model: ollama pull smollm:1.7b ● agents\_demo.py (file should have two agents + finalizer, strict JSON, no domain hardcoding)

## **Deliverables:**

1. **Command used (the exact python agents\_demo.py line).**

spartan@MLK-SCS-M7J3NJ9HTV 236 % python3 agents\_demo.py --model phi3:mini --title "Generative Adversarial Networks (GANs)" --content "Generative Adversarial Networks, or GANs, are a class of machine learning frameworks. Two neural networks, a generator and a discriminator, contest with each other in a zero-sum game. The generator creates fake data, while the discriminator tries to distinguish it from real data. This process improves both networks." --email "you@sjsu.edu" –strict

A computer screen with white text

AI-generated content may be incorrect.

1. **Console screenshot(s) showing:  
   ○ Planner output**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

○ **Reviewer output**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

○ **Finalized Output**

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**3.Short Answers (1 – 2 lines each):**

**Q1. Final 3 tags:**

1. "Generative Adversarial Networks (GAN)"
2. "Machine Learning"
3. "Artificial Intelligence"

**Q2. Final summary (15 words):**

"GAN, a deep learning technique where generator and discriminator networks compete to produce realistic data."

**Q3. Did Reviewer change anything?**

Yes - The Reviewer refined tags to be more specific and improved the summary for better technical accuracy.

**Q4. Explanation of the each step in your own words**

## **Tasks:**

1. Set up Ollama and models – phi3:mini model.
2. Created agents\_demo.py - A complete agentic AI system with:

* Planner Agent: Analyzes content and generates initial tags and summary
* Reviewer Agent: Reviews and improves the Planner's output
* Finalizer: Combines the best elements from both agents
* Publish Package: Creates the final structured output

1. Integrated Langchain with Ollama - Uses ChatOllama for local LLM inference