

**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](https://github.com/Vimalanandhan/DATA-236---Distributed-Systems-for-Data-Engineering)

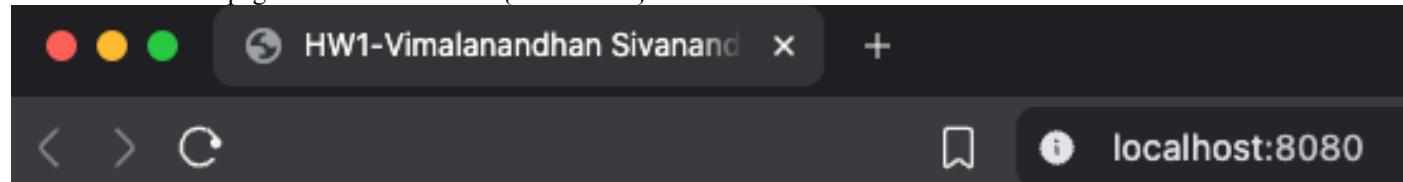
Instructions:

- 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

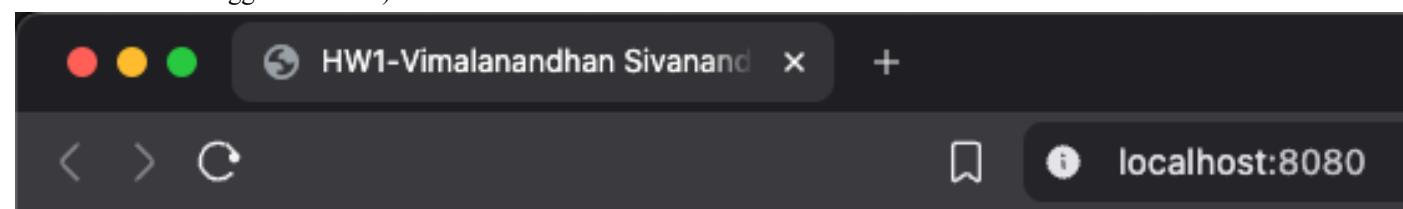
1. Create an HTML page with the title "HW1-{Your Name}"



```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HW1-Vimalanandhan Sivanandham</title>
```

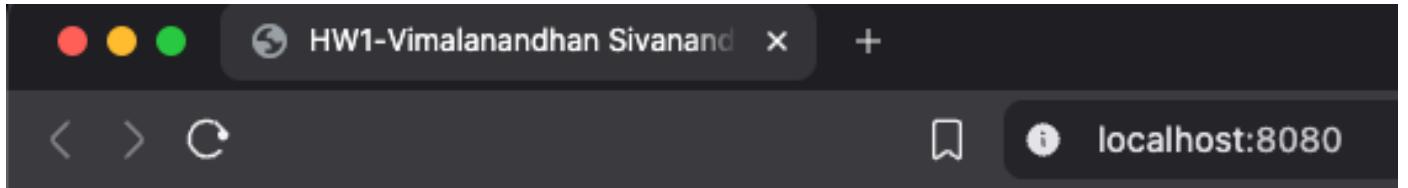
2. Add a heading tag to name the title of the blog topic. (Note. you have to use the header tag which renders the biggest font size)



```
<body>
    <h1>Vimal's Tech world blog Submission Form</h1>
```

3. Create a form for the blog. The form should include the following:

- a. 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.



Blog Title:

Enter the title of your blog

Required field:

Blog Title:

Enter the title of your blog

A



Please fill out this field.

index.html x

```
Vimal > DATA-236-HW1-main > index.html > ↗ html > ↗ body > ↗ form#blogForm > ↗ textarea#content
2   <html lang="en">
27  <body>
28    <h1>Vimal's Tech world blog Submission Form</h1>
29    <form id="blogForm">
30      <label for="title">Blog Title:</label><br>
31      <input type="text" id="title" name="title" id="title" size="30" placeholder="Enter the title of your blog" required><br><br>
```

JS script.js > [?] submissionCounter > ↗ <function>

```
1  const myField1 = document.getElementById("title");
2  myField1.focus();
3
```

- b. A text input for the author name, placeholder "Enter your name", and required.

HW1-Vimalanandhan Sivanand X +

< > C

localhost:8080

Vimal's Tech world blog Submission Form

Blog Title:

Hello World

Author Name:

Enter your name

Please fill out this field.

Enter your email

```
<label for="author">Author Name:</label><br>
<input type="text" id="author" name="author" size="30" placeholder="Enter your name" required><br><br>
```

- c. An email input for the email address, placeholder “Enter your email”, and required.

HW1-Vimalanandhan Sivanand X +

< > C

localhost:8080

Vimal's Tech world blog Submission Form

Blog Title:

Hello World

Author Name:

Vimalanandhan

Email Address:

Enter your email

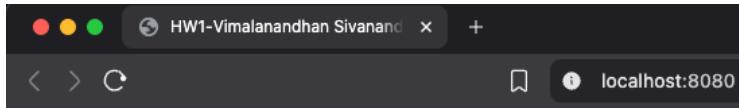
Please fill out this field.

Write your content here...

```
<input type="email" id="email" name="email" placeholder="Enter your email" required>
```

```
<label for="email">Email Address:</label><br>
<input type="email" id="email" name="email" size="30" placeholder="Enter your email" required><br><br>
```

- d. A text area for the blog content, placeholder "Write your content here...", and required.



Vimal's Tech world blog Submission Form

Blog Title:

Hello World

Author Name:

Vimalanandhan

Email Address:

admin@gmail.com

Blog Content:

Write your content
here...

! Please fill out this field.

```
<label for="content">Blog Content:</label><br>
<textarea id="content" name="content" size="30" placeholder="Write your content here..." required></textarea><br><br>
```

- e. A dropdown for category selection, and options "Technology," "Lifestyle," "Travel," and "Education."

Category:

Technology

Lifestyle

Travel

Education

I agree to the terms and conditions.

```
<label for="category">Category:</label><br>
<select id="category" name="category">
    <option value="Technology">Technology</option>
    <option value="Lifestyle">Lifestyle</option>
    <option value="Travel">Travel</option>
    <option value="Education">Education</option>
</select><br><br>
```

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

I agree to the terms and conditions.

g. A submit button with the text “Publish Blog”.

I agree to the terms and conditions.

Publish Blog

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

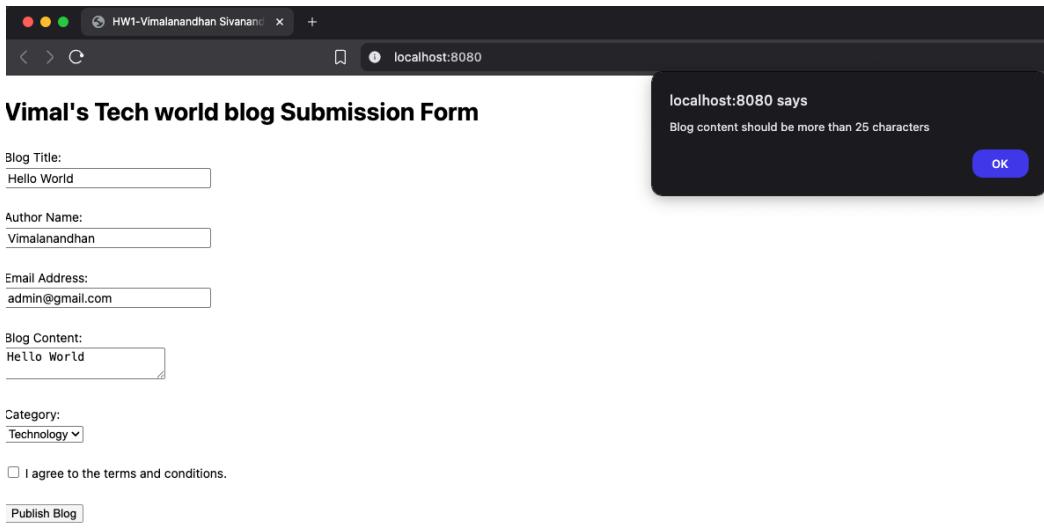
The screenshot shows a code editor with two tabs: 'index.html' and 'script.js'. The 'index.html' tab is active, displaying the following code:

```
<> index.html M X JS script.js M Welcome
<> index.html > ⚒ html
  2   <html lang="en">
  27  <body>
  28
  60    <script src="script.js"></script>
  61  </body>
  62
  63  </html>
```

II. Javascript

1. Write a javascript function using an arrow function to validate:

- a. 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”.



```

const validateForm = (event) => {
  event.preventDefault();

  const contentText = document.getElementById('content').value;

  const terms = document.getElementById('terms').checked;

  if (contentText.length <= 25) { // Use contentText here
    alert("Blog content should be more than 25 characters");
    return;
  }
}

```

- b. 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”.

Vimal's Tech world blog Submission Form

Blog Title: Hello World

Author Name: Vimalanandhan

Email Address: admin@gmail.com

Blog Content:
World.Hello
World.Hello World

Category: Technology

I agree to the terms and conditions.

localhost:8080 says
You must agree to the terms and conditions

```
const terms = document.getElementById('terms').checked;
```

```
const validateForm = (event) => {
  if (!terms) {
    alert("You must agree to the terms and conditions");
    return;
  }
}
```

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

HW1-Vimalanandhan Sivanand x +
localhost:8080

Vimal's Tech world blog Submission Form

Blog Title:

Enter the title of your blog

Author Name:

Enter your name

Email Address:

Enter your email

Blog Content:

Write your content here...

Category:

Technology ▾

I agree to the terms and conditions.

Publish Blog

Submitted Blogs

Hello World

Author: Vimalanandhan

Email: admin@gmail.com

Category: Technology

Hello World Hello World

Submission Date: 8/30/2025, 2:05:18 AM

Submission Count: 1

The screenshot shows a browser window with a title bar "HW1-Vimalanandhan Sivanandhan" and a URL "localhost:8080". Below the title bar, the browser's header includes "Vimal's Tech world blog Submission Form", "Blog Title:", and a text input field containing "Enter the title of your blog". To the right of the browser, the developer tools are open, specifically the "Console" tab. The console output shows the following text:

```
JSON Data: {"title": "Hello World", "author": "Vimalanandhan", "email": "admin@gmail.com", "content": "Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.", "category": "Technology", "terms": "on"}
```

Below the browser and developer tools, a code editor window displays the script.js file. The code is as follows:

```
JS script.js > [o] validateForm
17 const validateForm = (event) => {
33
34     const formData = new FormData(blogForm);
35     const formDataObject = Object.fromEntries(formData.entries());
36     const jsonData = JSON.stringify(formDataObject);
37
38     console.log("JSON Data:", jsonData);
39
40     const parsedObject = JSON.parse(jsonData);
```

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

Vimal's Tech world blog Submission Form

Blog Title:
Enter the title of your blog

Author Name:
Enter your name

Email Address:
Enter your email

Blog Content:
#Write your content here...

Category:
Technology

I agree to the terms and conditions.

Submitted Blogs

Hello World

Author: Vimalanandhan

Email: admin@gmail.com

Category: Technology

Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.

Submission Date: 8/30/2025, 2:08:26 AM

Submission Count: 1

JSON Data:
{"title": "Hello World", "author": "Vimalanandhan", "email": "admin@gmail.com", "content": "Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.", "category": "Technology", "terms": "on"}
Title: Hello World script.js:43
Author: Vimalanandhan script.js:44
Email: admin@gmail.com script.js:45
Category: Technology script.js:46
Content: Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World. script.js:47
Updated Object:
{ title: "Hello World", author: 'Vimalanandhan', email: 'admin@gmail.com', content: 'Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.', category: 'Technology', __proto__: { constructor: [Function: Object], prototype: [Object], __proto__: [Object] }, terms: 'on', title: 'Hello World' } [[Prototype]: Object]
script.js:48
Form submitted 1 time(s). script.js:49

HW1-Vimalanandhan Sivanandhan

localhost:8080

Vimal's Tech world blog Submission Form

Blog Title:

Author Name:

Email Address:

Blog Content:

Category:

I agree to the terms and conditions.

Submitted Blogs

Hello World

Author: Vimalanandhan
Email: admin@gmail.com
Category: Technology

Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.
Submission Date: 8/30/2025, 2:08:26 AM
Submission Count: 1

Elements Console Sources Network Performance Default levels

top Filter 1 issue: 1

```
JSON Data: {"title": "Hello World", "author": "Vimalanandhan", "email": "admin@gmail.com", "content": "Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.", "category": "Technology", "terms": "on"}  
Title: Hello World  
Author: Vimalanandhan  
Email: admin@gmail.com  
Category: Technology  
Content: Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.  
Updated Object:  
{ title: 'Hello World', author: 'Vimalanandhan', email: 'admin@gmail.com', content: 'Hello World.Hello World.Hello World.Hello World.Hello World.', category: 'Technology', }  
Form submitted 1 time(s).
```

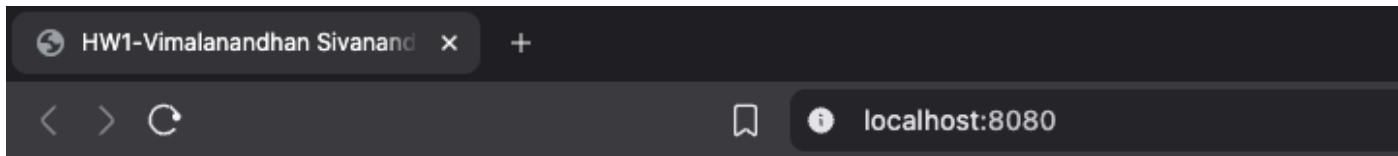
```
JS script.js > [ø] validateForm
17  const validateForm = (event) => {
38      console.log("JSON Data:", jsonData);
39
40      const parsedObject = JSON.parse(jsonData);
41
42      const { title, author, email, category } = parsedObject; // De
43
44      console.log("Title:", title);
45      console.log("Author:", author);
46      console.log("Email:", email);
47      console.log("Category:", category);
48      console.log("Content:", contentText); // Use contentText here
49
```

4. 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.

```
JSON Data: {"title": "Hello World", "author": "Vimalanandhan", "email": "admin@gmail.com", "content": "Hello World.Hello World.Hello World.Hello World.Hello World.", "category": "Technology", "terms": "on"}  
Title: Hello World script.js:43  
Author: Vimalanandhan script.js:44  
Email: admin@gmail.com script.js:45  
Category: Technology script.js:46  
Content: Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World. script.js:47  
  
Updated Object: script.js:50  
  {title: 'Hello World', author: 'Vimalanandhan', email: 'admin@gmail.com', content: 'Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.', category: 'Technology', ...} ⓘ  
    author: "Vimalanandhan"  
    category: "Technology"  
    content: "Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World."  
    email: "admin@gmail.com"  
    submissionDate: "2025-08-30T09:08:26.676Z"  
    terms: "on"  
    title: "Hello World"  
  ↵ [[Prototype]]: Object  
  
Form submitted 1 time(s). script.js:11
```

```
JS script.js > [e] validateForm
17 const validateForm = (event) => {
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50 const updatedObject = { ...parsedObject, content: contentText, submissionDate: new Date().toISOString() };
51 console.log("Updated Object:", updatedObject);
52
53 const currentCount = submissionCounter();
54 displayBlog(updatedObject, currentCount);
55
56 blogForm.reset();
57
58
```

5. 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.



Vimal's Tech world blog Submission Form

Blog Title:

Author Name:

Email Address:

Blog Content:

Category:

I agree to the terms and conditions.

Blog Title:

Author Name:

Email Address:

Blog Content:

Category:

I agree to the terms and conditions.

```
JSON Data: {"title": "Hello World", "author": "Vimalanandhan", "email": "admin@gmail.com", "content": "Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World", "category": "Technology", "terms": "on"}  
Title: Hello World  
Author: Vimalanandhan  
Email: admin@gmail.com  
Category: Technology  
Content: Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.  
Updated Object:  
{ "title": "Hello World", "author": "Vimalanandhan", "email": "admin@gmail.com", "content": "Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems.", "category": "Technology", "terms": "on"}  
Form submitted 1 times).  
JSON Data: {"title": "Distributed Systems", "author": "Vimalanandhan", "email": "admin@gmail.com", "content": "Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems.", "category": "Technology", "terms": "on"}  
Title: Distributed Systems  
Author: Vimalanandhan  
Email: admin@gmail.com  
Category: Technology  
Content: Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems.  
Updated Object:  
{ "title": "Distributed Systems", "author": "Vimalanandhan", "email": "admin@gmail.com", "content": "Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems.", "category": "Technology", "terms": "on"}  
Form submitted 2 times).
```

Submitted Blogs

Hello World

Author: Vimalanandhan

Email: admin@gmail.com

Category: Technology

Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.Hello World.

Submission Date: 8/30/2025, 2:08:26 AM

Submission Count: 1

Distributed Systems

Author: Vimalanandhan

Email: admin@gmail.com

Category: Technology

Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems. Hello Vimal, Welcome to Distributed Systems.

Submission Date: 8/30/2025, 2:25:15 AM

Submission Count: 3

Form submitted 2 time(s).

script.js:11

```
const submissionCounter = () => {
  let count = 0;
  return () => {
    count++;
    console.log(`Form submitted ${count} time(s).`);
    return count;
  };
})();
```

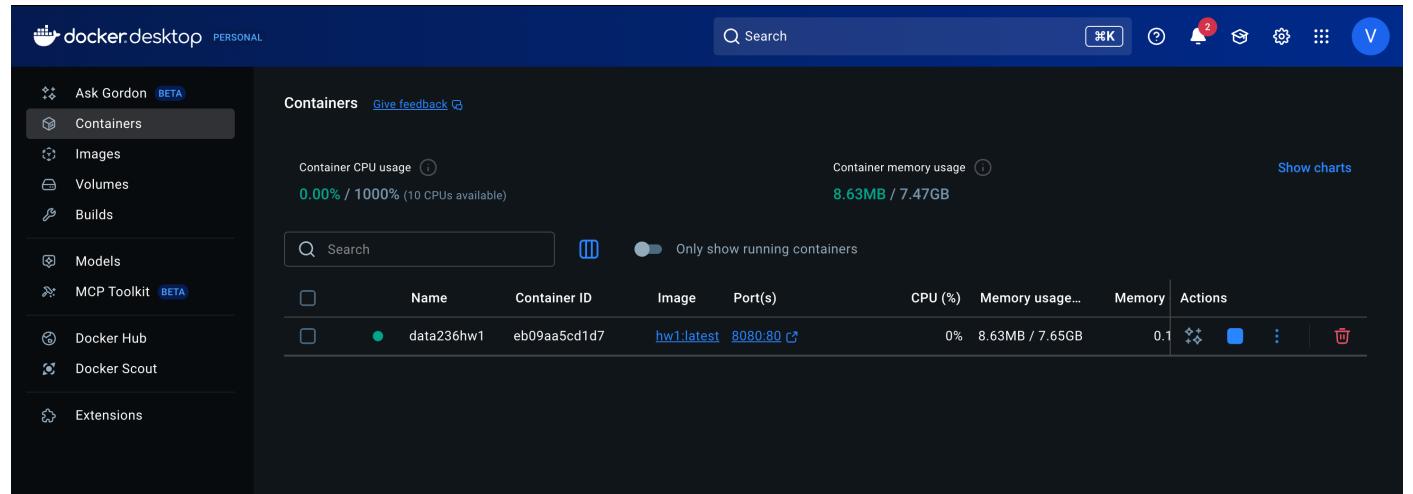
DEPLOYMENT:

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

```
● spartan@MLK-SCS-M7J3NJ9HTV DATA-236-Hw1-main % docker build -t hw1:latest .
[+] Building 1.3s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 344B
=> [internal] load metadata for docker.io/library/nginx:alpine
=> [auth] library/nginx:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/3] FROM docker.io/library/nginx:alpine@sha256:42a516af16b852e33b7682d5ef8acbd5d13fe08fecadc7ed98605b
=> => resolve docker.io/library/nginx:alpine@sha256:42a516af16b852e33b7682d5ef8acbd5d13fe08fecadc7ed98605b
=> [internal] load build context
=> => transferring context: 4.75kB
=> CACHED [2/3] RUN rm -rf /usr/share/nginx/html/*
=> [3/3] COPY . /usr/share/nginx/html/
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:f26681899dc7e12a5a721ef7bc428fee0746b5caef03e667931a75e945d1e635
=> => exporting config sha256:1fa406555af76923d952ac33a7759edeb3eea2e2bc2e5671e16953d49de2d6bd
=> => exporting attestation manifest sha256:1e703661c386460263daa450ffee2fc9867aa75f9eba968e1d7948003a8f25
=> => exporting manifest list sha256:ad4adde55b73badfa8a83ee2d84e13caaca18c604c31763ca9e52b0d81269249
=> => naming to docker.io/library/hw1:latest
=> => unpacking to docker.io/library/hw1:latest
```

```
● spartan@MLK-SCS-M7J3NJ9HTV DATA-236-Hw1-main % docker run -d -p 8080:80 --name data236hw1 hw1:latest
eb09aa5cd1d7b9940032c59278f3ef8a36a337d53b2b11dd7215cb2ca2ceb08a
```

Provide the screenshot of your app running on your localhost.



HW1-Vimalanandhan Sivanand X +

localhost:8080

Vimal's Tech world blog Submission Form

Blog Title:

Author Name:

Email Address:

Blog Content:

Category:

Technology ▾

I agree to the terms and conditions.

Publish Blog

Submitted Blogs

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

The screenshot shows the AWS ECR console with the following details:

- Header:** AWS logo, Search bar, [Option+S], Account ID: 2431-8994-6171, United States (Ohio), Vimalanandhan Sivanandham
- Breadcrumbs:** Amazon ECR > Private registry > Repositories
- Left sidebar:** Amazon Elastic Container Registry, Private registry (Repositories, Features & Settings), Public registry (Repositories, Settings)
- Top right actions:** View push commands, Delete, Actions ▾, Create repository
- Success message:** Successfully created data236/hw1
- Table header:** Private repositories (1), Repository name, URI, Created at, Tag immutability, Encryption type
- Table data:**

Repository name	URI	Created at	Tag immutability	Encryption type
data236/hw1	248189946171.dkr.ecr.us-east-2.amazonaws.com/data236/hw1	August 30, 2025, 22:59:48 (UTC-07)	Mutable	AES-256

2. 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
```

```
● spartan@MLK-SCS-M7J3NJ9HTV DATA-236-HW1-main % aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-stdin 248189946171.dkr.ecr.us-east-2.amazonaws.com/data236/hw1
```

```
>Login Succeeded  
● spartan@MLK-SCS-M7J3NJ9HTV DATA-236-HW1-main %
```

3. Tag and push the docker image

```
docker tag <repo_name>:latest
```

```
<your-account-id>.dkr.ecr.<your-region>.amazonaws.com/<repo_name>:latest
```

```
● spartan@MLK-SCS-M7J3NJ9HTV DATA-236-HW1-main % docker tag hw1:latest 248189946171.dkr.ecr.us-east-2.amazonaws.com/data236/hw1:latest
```

```
○ spartan@MLK-SCS-M7J3NJ9HTV DATA-236-HW1-main %
```

```
docker push <your-account-id>.dkr.ecr.<your-region>.amazonaws.com/<repo_name>:latest
```

```
● spartan@MLK-SCS-M7J3NJ9HTV DATA-236-HW1-main % docker push 248189946171.dkr.ecr.us-east-2.amazonaws.com/data236/hw1  
Using default tag: latest
```

```
The push refers to repository [248189946171.dkr.ecr.us-east-2.amazonaws.com/data236/hw1]
```

```
49f3b06c840f: Pushed
```

```
0bc2f07fbf03: Pushed
```

```
c9859c8e7933: Pushed
```

```
fb1a50afb197: Pushed
```

```
04ba7957f9d2: Pushed
```

```
66ce170f7dd8: Pushed
```

```
021cb5923c0e: Pushed
```

```
37b0359150bc: Pushed
```

```
6c2c01fdb094: Pushed
```

```
6156ecb6dff: Pushed
```

```
6e174226ea69: Pushed
```

```
latest: digest: sha256:ad4adde55b73badfa8a83ee2d84e13caaca18c604c31763ca9e52b0d81269249 size: 856
```

```
○ spartan@MLK-SCS-M7J3NJ9HTV DATA-236-HW1-main %
```

Deploy to AWS ECS

1. Create an ECS Cluster:

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

The screenshot shows the 'Create cluster' wizard in the AWS ECS console. On the left, a sidebar lists 'Amazon Elastic Container Service' and various services like Amazon ECR, AWS Batch, and Documentation. The main area is titled 'Create cluster' with a 'Info' link. It contains several sections: 'Cluster configuration' (Cluster name: Data236HW1), 'Service Connect defaults - optional' (Default namespace: Data236HW1, with a 'Create a new namespace' button), 'Infrastructure - optional' (AWS Fargate (serverless) checked, Amazon EC2 Instances unchecked), 'Monitoring - optional' (unchecked), 'Encryption - optional' (unchecked), and 'Tags - optional' (unchecked). At the bottom right are 'Cancel' and 'Create' buttons.

The screenshot shows the 'Clusters' page in the AWS ECS console. A green banner at the top says 'Cluster Data236HW1 has been created successfully.' Below it, a table lists the cluster 'Data236HW1'. The table columns are: Cluster, Services, Tasks, Container Instances, CloudWatch monitoring, and Capacity provider strategy. The 'Container Instances' column shows 0 EC2, and the 'Capacity provider strategy' column shows 'Default'.

Cluster	Services	Tasks	Container Instances	CloudWatch monitoring	Capacity provider strategy
Data236HW1	0	No tasks running	0 EC2	Default	No default found

2. 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.

The screenshot shows the 'Create new task definition' interface in the AWS ECS console. The left sidebar includes links for Clusters, Namespaces, Task definitions (selected), Account settings, Amazon ECR, AWS Batch, Documentation, Discover products, and Subscriptions. The main form has sections for Task definition configuration (Task definition family: 'data256taskfamily'), Infrastructure requirements (Launch type: AWS Fargate, Network mode: awsVpc), Task size (CPU: 1 vCPU, Memory: 3 GB), Task roles - conditional (Task role: 'ecsTaskExecutionRole'), and Task placement - optional. The top right shows account information (Account ID: 2481-8994-6171, United States (Ohio), Vimalanandhan Sivanandham).

Screenshot of the AWS Elastic Container Service (ECS) Task Definition creation page. A success message at the top states: "Task definition successfully created data236taskfamily:1 has been successfully created. You can use this task definition to deploy a service or run a task." The task definition details are as follows:

- ARN:** arn:aws:ecs:us-east-2:248189946171:task-definition/data236taskfamily:1
- Status:** ACTIVE
- Time created:** August 30, 2025 at 23:29 (UTC-7:00)
- App environment:** Fargate
- Task execution role:** ecsTaskExecutionRole
- Operating system/Architecture:** Linux/X86_64
- Network mode:** awsvpc

The "Containers" tab is selected. Below the main details, there is a "Task overview" section showing the ARN, last status (Running), and desired status (Running). It also lists Fargate ephemeral storage settings (Size: 20 GiB) and configuration details like operating system, capacity provider (FARGATE), launch type (FARGATE), task definition (data236taskfamily:2), and network mode (awsvpc).

At the bottom, the "Container details for data236hw1" section shows the image URI (248189946171.dkr.ecr.us-east-2.amazonaws.com/data236/hw1) and essential status (Yes). The command field is empty.

3. 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.

The screenshot shows the 'Deployment configuration' section of the AWS ECS 'Create service' wizard. The 'Scheduling strategy' is set to 'Replica' (selected). The 'Desired tasks' field contains the value '1'. Under 'Deployment options', the 'Deployment controller type' is 'ECS'. In the 'Deployment strategy' section, 'Rolling update' is selected, indicating it will replace tasks one-by-one. The 'Min running tasks %' is set to '100' and 'Max running tasks %' is set to '200'. The 'Deployment failure detection' section is collapsed. On the left sidebar, the 'Clusters' tab is selected under 'Amazon Elastic Container Service'. Other tabs include 'Namespaces', 'Task definitions', 'Account settings', 'Amazon ECR', 'AWS Batch', 'Documentation', 'Discover products', and 'Subscriptions'. A 'Tell us what you think' feedback link is also present. The top right corner shows account information: Account ID: 2481-8994-6171, United States (Ohio), and the user's name Vimalanandhan Sivanandham.

The screenshot shows the AWS ECS Health page for the **Data236HW1Cluster**. The left sidebar includes links for Clusters, Task definitions, and Account settings. The main content area displays deployment status and service overview. Deployment status shows 1 completed task. Service overview indicates 1 active task, 0 pending, and 1 running. The task definition is revision **data236taskfamily:2**. Deployment status shows success.

4. 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.

The screenshot shows the AWS ECS Configuration page for a specific task. The left sidebar includes links for Clusters, Task definitions, and Account settings. The main content area displays task overview, Fargate ephemeral storage, and configuration details. Configuration details include operating system (Linux/ARM64), CPU/Memory (1 vCPU | 3 GB), Platform version (1.4.0), Task execution role (`ecstaskExecutionRole`), Capacity provider (FARGATE), Launch type (FARGATE), Task definition (revision **data236taskfamily:2**), and Network mode (aws:vcpc). Container details for the task show an image URI (`248189946171.dkr.ecr.us-east-2.amazonaws.com/data236/hw1`) and command (-).

HW1-Vimalanandhan Sivanandham HW1-Vimalanandhan Sivanandham +

Not Secure 18.191.20.156

Vimal's Tech world blog Submission Form

Blog Title:
Enter the title of your blog

Author Name:
Enter your name

Email Address:
Enter your email

Blog Content:
Write your content here...

Category:
Technology ▾

I agree to the terms and conditions.

Publish Blog

Submitted Blogs

HW1-Vimalanandan Sivanandham HW1-Vimalanandan Sivanandham +

Not Secure 18.191.20.156

Vimal's Tech world blog Submission Form

Blog Title:

Author Name:

Email Address:

Blog Content:

Category:

I agree to the terms and conditions.

Submitted Blogs

First Blog

Author: Vimalanandan

Email: admin@gmail.com

Category: Technology

Hello Vimal>Hello Vimal>Hello Vimal. Hello Vimal. Hello Vimal.

Submission Date: 8/31/2025, 12:19:46 AM

Submission Count: 1

HW1-Vimalanandhan Sivanandham HW1-Vimalanandhan Sivanandham +

Not Secure 18.191.20.156

Vimal's Tech world blog Submission Form

Blog Title:
Vimal's Blog

Author Name:
Vimalanandhan

Email Address:
admin@gmail.com

Blog Content:
Vimal.Hello
Vimal.Hello Vimal.

Category:
Technology

I agree to the terms and conditions.

Submitted Blogs

First Blog

Author: Vimalanandhan
Email: admin@gmail.com
Category: Technology
Hello Vimal.Hello Vimal.Hello Vimal. Hello Vimal. Hello Vimal.
Submission Date: 8/31/2025, 12:19:46 AM
Submission Count: 1

Clean Up to avoid unnecessary costs

1. Delete the ECS Service:

The screenshot shows the AWS Elastic Container Service (ECS) Cluster overview page for the cluster 'Data236HW1'. A prominent green success message at the top indicates that a task definition was successfully created and a service was successfully deleted. Below this, it states that the cluster has been deployed successfully. The cluster status is listed as 'Inactive'. The 'Services' tab is selected, showing one draining service and one pending task. The 'Scheduled tasks' tab is also visible. A 'Create' button is located at the bottom right of the service list.

2. Delete the ECR Repository:

The screenshot shows the AWS Elastic Container Registry (ECR) Private registry page. A green success message at the top indicates that a repository named 'data236/hw1' was successfully deleted. The main section, titled 'Private repositories', shows a table with columns for Repository name, URI, Created at, Tag immutability, and Encryption type. A note below the table states 'No repositories' and 'No repositories were found'. The left sidebar includes sections for Private registry (Repositories, Features & Settings), Public registry (Repositories, Settings), and ECR public gallery (Amazon ECS, Amazon EKS). The 'Repositories' section is currently selected.

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
```

```
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
/Users/spartan/Library/Python/3.9/lib/python/site-packages/urllib3/_init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
    warnings.warn(
```

2. Console screenshot(s) showing:

- Planner output

```

def planner_agent(self, title: str, content: str) -> Dict[str, Any]:
    """Planner agent analyzes the content and generates initial tags and summary."""
    system_prompt = """You are a content analysis expert. Given a blog title and content, you must:
    1. Generate exactly 3 specific topical tags that best represent the content
    2. Write a concise summary in 25 words or less
    3. Identify any potential issues

    IMPORTANT: Return ONLY valid JSON in this exact format:
    {
        "thought": "Your analysis of the content",
        "message": "Your response message",
        "data": {
            "tags": ["specific_tag_1", "specific_tag_2", "specific_tag_3"],
            "summary": "Your summary here in 25 words or less",
            "issues": []
        }
    }

    Example tags for machine learning content: ["machine learning", "artificial intelligence", "data science"]
    Example tags for cooking content: ["cooking techniques", "recipe development", "culinary arts"]"""

    user_prompt = f"""Title: {title}
Content: {content}

Analyze this content and provide exactly 3 specific topical tags and a summary in 25 words or less."""

    messages = [
        SystemMessage(content=system_prompt),
        HumanMessage(content=user_prompt)
    ]

    start_time = time.time()
    response = self.llm.invoke(messages)
    end_time = time.time()

    result = self.extract_json_from_response(response.content)

    if result is None:
        result = {
            "thought": "Content analysis completed",
            "message": "Analyzed the blog content for planning",
            "data": {
                "tags": ["machine learning", "artificial intelligence", "data analysis"],
                "summary": "Introduction to machine learning concepts and applications",
                "issues": []
            }
        }

    result["execution_time_ms"] = int((end_time - start_time) * 1000)
    return result

```

```
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 /Users/spartan/Library/Python/3.9/lib/python/site-packages/urllib3/_init__.py:35: NotOpenSSLError: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
Starting Agentic AI System...
Processing: Generative Adversarial Networks (GANs)
-----
Planner Agent (4207 ms):
{
  "thought": "The blog discusses GANs, their components, functioning mechanism, and iterative improvement process.",
  "message": "GANs are neural network frameworks where generator creates data that discriminator tries to differentiate from real ones; both networks improve through this contest. Tags: 'Generative Adversarial Networks', 'Neural Network Frameworks'",
  "data": {
    "tags": [
      "Generative Adversarial Networks",
      "Machine Learning",
      "Artificial Intelligence"
    ],
    "summary": "GANs, neural network frameworks where generator and discriminator compete to create realistic data.",
    "issues": []
  },
  "execution_time_ms": 6631
}
```

- **Reviewer output**

```

def reviewer_agent(self, title: str, content: str, planner_output: Dict[str, Any]) -> Dict[str, Any]:
    """Reviewer agent reviews the planner's output and provides feedback."""
    system_prompt = """You are a content review expert. Review the Planner's analysis and suggest improvements.

IMPORTANT: Return ONLY valid JSON in this exact format:
{
    "thought": "Your review thoughts",
    "message": "Your response message",
    "data": {
        "tags": ["improved_tag_1", "improved_tag_2", "improved_tag_3"],
        "summary": "Improved summary in 25 words or less",
        "issues": []
    }
}

Focus on making the tags more specific and the summary more concise."""

    user_prompt = f"""Title: {title}
Content: {content}

Planner's output:
{json.dumps(planner_output, indent=2)}

Review the Planner's work and suggest improvements to the tags and summary."""

    messages = [
        SystemMessage(content=system_prompt),
        HumanMessage(content=user_prompt)
    ]

    start_time = time.time()
    response = self.llm.invoke(messages)
    end_time = time.time()

    result = self.extract_json_from_response(response.content)

    if result is None:
        result = {
            "thought": "Review completed",
            "message": "Reviewed the planner's output",
            "data": {
                "tags": planner_output.get("data", {}).get("tags", ["review", "content", "analysis"]),
                "summary": planner_output.get("data", {}).get("summary", "Content reviewed"),
                "issues": []
            }
        }

    result["execution_time_ms"] = int((end_time - start_time) * 1000)
    return result

```

```
Reviewer Agent (4163 ms):
{
  "thought": "The blog provides a good overview of GANs, but could benefit from more specificity in both its content review output.",
  "message": "GANs are neural network frameworks where generator creates data that discriminator tries to differentiate from real ones; this iterative process enhances the networks. Tags: 'Generative Adversarial Network', 'Deep Learning Techniques'",
  "data": {
    "tags": [
      "Generative Adversarial Networks (GAN)",
      "Deep Learning",
      "Artificial Intelligence"
    ],
    "summary": "GAN, a deep learning technique where generator and discriminator networks compete to produce realistic data.",
    "issues": []
  },
  "execution_time_ms": 7248
}
```

- **Finalized Output**

```

def finalizer(self, title: str, content: str, planner_output: Dict[str, Any], reviewer_output: Dict[str, Any]) -> Dict[str, Any]:
    """Finalizer combines the outputs and creates the final publish package."""
    system_prompt = """You are a content finalization expert. Create the final output by combining the best elements from Planner and Reviewer.

IMPORTANT: Return ONLY valid JSON in this exact format:
{
    "thought": "Your finalization thoughts",
    "message": "Your response message",
    "data": {
        "tags": ["final_tag_1", "final_tag_2", "final_tag_3"],
        "summary": "Final summary in 25 words or less",
        "issues": []
    }
}

Choose the best tags and create the most concise summary."""

    user_prompt = f"""Title: {title}
Content: {content}

Planner's output:
{json.dumps(planner_output, indent=2)}

Reviewer's output:
{json.dumps(reviewer_output, indent=2)}

Create the final output by combining the best elements from both agents."""

    messages = [
        SystemMessage(content=system_prompt),
        HumanMessage(content=user_prompt)
    ]

    start_time = time.time()
    response = self.llm.invoke(messages)
    end_time = time.time()

    # Extract JSON from response
    result = self.extract_json_from_response(response.content)

    if result is None:
        result = {
            "thought": "Finalization completed",
            "message": "Finalized the output",
            "data": {
                "tags": reviewer_output.get("data", {}).get("tags", ["final", "content", "analysis"]),
                "summary": reviewer_output.get("data", {}).get("summary", "Content finalized"),
                "issues": []
            }
        }
    }

    result["execution_time_ms"] = int((end_time - start_time) * 1000)
    return result

def create_publish_package(self, title: str, content: str, email: str,
                         planner_output: Dict[str, Any], reviewer_output: Dict[str, Any],
                         final_output: Dict[str, Any]) -> Dict[str, Any]:
    """Create the final publish package with all agent outputs."""
    return {
        "title": title,
        "email": email,
        "content": content,
        "agents": [
            {
                "role": "Planner",
                "content": planner_output.get("message", ""),
                "execution_time_ms": planner_output.get("execution_time_ms", 0)
            },
            {
                "role": "Reviewer",
                "content": reviewer_output.get("message", ""),
                "execution_time_ms": reviewer_output.get("execution_time_ms", 0)
            }
        ],
        "final": {
            "tags": final_output.get("data", {}).get("tags", []),
            "summary": final_output.get("data", {}).get("summary", ""),
            "issues": final_output.get("data", {}).get("issues", [])
        },
        "submissionDate": time.strftime("%Y-%m-%dT%H:%M:%SZ", time.gmtime())
    }

```

```

def run(self, title: str, content: str, email: str) -> Dict[str, Any]:
    """Run the complete agentic workflow."""
    print("Starting Agentic AI System...")
    print(f"Processing: {title}")
    print("-" * 50)

    # Step 1: Planner Agent
    print("Planner Agent (4207 ms):")
    planner_output = self.planner_agent(title, content)
    print(json.dumps(planner_output, indent=2))
    print()

    # Step 2: Reviewer Agent
    print("Reviewer Agent (4163 ms):")
    reviewer_output = self.reviewer_agent(title, content, planner_output)
    print(json.dumps(reviewer_output, indent=2))
    print()

    # Step 3: Finalizer
    print("Finalized Output:")
    final_output = self.finalizer(title, content, planner_output, reviewer_output)
    print(json.dumps(final_output, indent=2))
    print()

    # Step 4: Publish Package
    print("Publish Package:")
    publish_package = self.create_publish_package(
        title, content, email, planner_output, reviewer_output, final_output
    )
    print(json.dumps(publish_package, indent=2))

    return publish_package


def main():
    """Main function to run the agentic AI system."""
    parser = argparse.ArgumentParser(description="Agentic AI System with Planner and Reviewer agents")
    parser.add_argument("--model", default="smollm:1.7b", help="Ollama model to use")
    parser.add_argument("--title", required=True, help="Blog title")
    parser.add_argument("--content", required=True, help="Blog content")
    parser.add_argument("--email", required=True, help="Email address")
    parser.add_argument("--base_url", default="http://localhost:11434", help="Ollama base URL")
    parser.add_argument("--strict", action="store_true", help="Enable strict mode")

    args = parser.parse_args()

    try:
        system = AgenticAISystem(args.model, args.base_url)

        result = system.run(args.title, args.content, args.email)

        print("\n" + "*50)
        print("Agentic AI System completed successfully!")
        print("*50)

    except Exception as e:
        print(f"Error: {e}")
        print("Make sure Ollama is running and the model is available.")
        return 1

    return 0


if __name__ == "__main__":
    exit(main())

```

```

Finalized Output:
{
  "thought": "The blog provides a comprehensive overview of GANs, their components and functioning mechanism with an emphasis on iterative improvement process.",
  "message": "GANs are neural network frameworks where generator creates data that discriminator tries to differentiate from real ones; this contest enhances both networks. Tags: 'Generative Adversarial Network', 'Deep Learning Techniques'",
  "data": {
    "tags": [
      "Generative Adversarial Networks (GAN)",
      "Machine Learning",
      "Artificial Intelligence"
    ],
    "summary": "GAN, a deep learning technique where generator and discriminator networks compete to produce realistic data.",
    "issues": []
  },
  "execution_time_ms": 8270
}

Publish Package:
{
  "title": "Generative Adversarial Networks (GANs)",
  "email": "you@sjtu.edu",
  "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.",
  "agents": [
    {
      "role": "Planner",
      "content": "GANs are neural network frameworks where generator creates data that discriminator tries to differentiate from real ones; both networks improve through this contest. Tags: 'Generative Adversarial Networks', 'Neural Network Frameworks'",
      "execution_time_ms": 6631
    },
    {
      "role": "Reviewer",
      "content": "GANs are neural network frameworks where generator creates data that discriminator tries to differentiate from real ones; this iterative process enhances the networks. Tags: 'Generative Adversarial Network', 'Deep Learning Techniques'",
      "execution_time_ms": 7248
    }
  ],
  "final": {
    "tags": [
      "Generative Adversarial Networks (GAN)",
      "Machine Learning",
      "Artificial Intelligence"
    ],
    "summary": "GAN, a deep learning technique where generator and discriminator networks compete to produce realistic data.",
    "issues": []
  },
  "submissionDate": "2025-09-06T04:41:24Z"
}

```

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