

Team Name: Elite

Project Title: Distributed Event Management System

The Distributed Event Management System is a fully functional application comprising three Flask-based microservices—Event Registration, Ticketing, and User Engagement—integrated with a React.js frontend and AWS DynamoDB for data storage. The backend services handle operations like event creation, ticket purchases, and user notifications, exposing REST APIs for interaction. These services run in Docker containers, ensuring modularity and scalability. The React frontend dynamically fetches and displays data, allowing users to view events, purchase tickets, and check notifications. The system was demonstrated by running the backend services locally, testing them using Postman, showcasing dynamic frontend functionality with npm start, and highlighting AWS DynamoDB as the data layer. Additionally, the locally deployed frontend was accessed globally, showcasing the system's distributed architecture and deployment capability.

Steps performed to execute the project:

For Backend:

Step 1: Set Up the AWS Environment

1. Create an AWS Account:

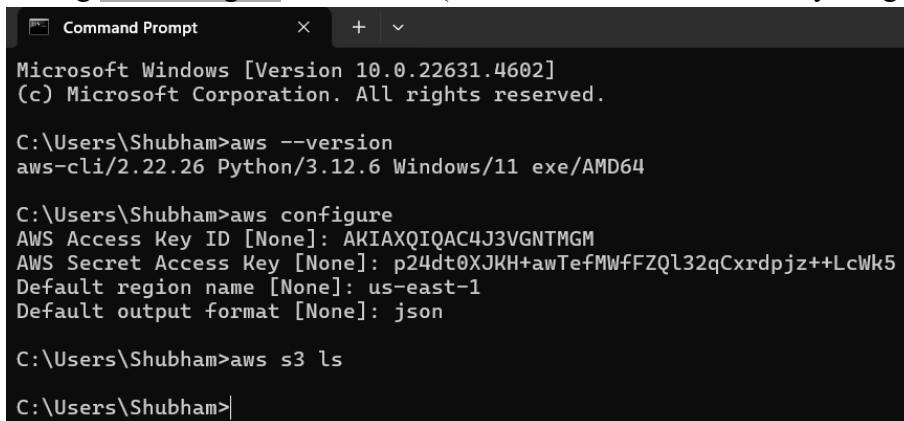
Sign up / Sign in at AWS.

Configure billing alerts to track costs.

2. Set Up AWS CLI:

Install the AWS CLI on your machine.

Configure it using `aws configure` command. (Provide access and secret keys, region).



```
Microsoft Windows [Version 10.0.22631.4602]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Shubham>aws --version
aws-cli/2.22.26 Python/3.12.6 Windows/11 exe/AMD64

C:\Users\Shubham>aws configure
AWS Access Key ID [None]: AKIAHQIQAC4J3VGNTMGH
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: json

C:\Users\Shubham>aws s3 ls

C:\Users\Shubham>
```

3. Create an EKS Cluster:

Use the AWS CLI to create an Elastic Kubernetes Service (EKS) cluster.

Use the default configuration to start with.

Install kubectl and configure it for your EKS cluster.

`aws eks update-kubeconfig --region <region> --name <cluster_name>`

Clusters (1) Info

[Delete](#)[Create cluster](#)

< 1 >

Cluster name ▲	Status ▼	Kubernetes version ▼	Support period	Upgrade policy ▼	Created
event-cluster	Active	1.30 Upgrade now	Standard support until July 28, 2025	Extended	January 1, 2025, 16:36 (UTC+00:00)

4. Create an S3 Bucket:

Use S3 for storing static assets like event banners or logs.

Create a bucket using the AWS CLI:

```
aws s3api create-bucket --bucket <bucket-name> --region <region>
```

```
PS C:\Users\Shubham> aws s3api create-bucket --bucket event-bucket-ucd2024 --region us-east-1
{
  "Location": "/event-bucket-ucd2024"
}

PS C:\Users\Shubham> aws s3 ls
2025-01-01 16:39:56 event-bucket-ucd2024/
PS C:\Users\Shubham>
```

Step 2: Design the Database Schema in DynamoDB

1. Set Up DynamoDB Tables:

```
C:\Users\Shubham>aws dynamodb create-table --table-name Events ^
More? --attribute-definitions AttributeName=EventID,AttributeType=S AttributeName=OrganizerID,AttributeType=S ^
More? --key-schema AttributeName=EventID,KeyType=HASH AttributeName=OrganizerID,KeyType=RANGE ^
More? --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 --region us-east-1
{
  "TableDescription": {
    "AttributeDefinitions": [
      {
        "AttributeName": "EventID",
        "AttributeType": "S"
      },
      {
        "AttributeName": "OrganizerID",
        "AttributeType": "S"
      }
    ],
    "TableName": "Events",
    "KeySchema": [
      {
        "AttributeName": "EventID",
        "KeyType": "HASH"
      },
      {
        "AttributeName": "OrganizerID",
        "KeyType": "RANGE"
      }
    ],
    "TableStatus": "CREATING",
    "CreationDateTime": "2025-01-01T16:50:31.882000+00:00",
    "ProvisionedThroughput": {
      "NumberOfDecreasesToday": 0,
      "ReadCapacityUnits": 5,
      "WriteCapacityUnits": 5
    },
    "TableSizeBytes": 0,
    "ItemCount": 0,
    "TableArn": "arn:aws:dynamodb:us-east-1:515966506771:table/Events",
    "TableId": "96d4fa67-e7b0-4134-b2ca-284ad9b7d098",
    "DeletionProtectionEnabled": false
  }
}
```

```

C:\Users\Shubham>aws dynamodb create-table --table-name Tickets ^
More? --attribute-definitions AttributeName=TicketID,AttributeType=S ^
More? --key-schema AttributeName=TicketID,KeyType=HASH ^
More? --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 --region us-east-1
{
  "TableDescription": {
    "AttributeDefinitions": [
      {
        "AttributeName": "TicketID",
        "AttributeType": "S"
      }
    ],
    "TableName": "Tickets",
    "KeySchema": [
      {
        "AttributeName": "TicketID",
        "KeyType": "HASH"
      }
    ],
    "TableStatus": "CREATING",
    "CreationDateTime": "2025-01-01T16:51:15.803000+00:00",
    "ProvisionedThroughput": {
      "NumberOfDecreasesToday": 0,
      "ReadCapacityUnits": 5,
      "WriteCapacityUnits": 5
    },
    "TableSizeBytes": 0,
    "ItemCount": 0,
    "TableArn": "arn:aws:dynamodb:us-east-1:515966506771:table/Tickets",
    "TableId": "cf876fef-0ad6-4707-9f53-355d579bb773",
    "DeletionProtectionEnabled": false
  }
}

```

```

C:\Users\Shubham>aws dynamodb create-table --table-name Notifications ^
More? --attribute-definitions AttributeName=UserID,AttributeType=S AttributeName=Timestamp,AttributeType=N ^
More? --key-schema AttributeName=UserID,KeyType=HASH AttributeName=Timestamp,KeyType=RANGE ^
More? --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 --region us-east-1
{
  "TableDescription": {
    "AttributeDefinitions": [
      {
        "AttributeName": "Timestamp",
        "AttributeType": "N"
      },
      {
        "AttributeName": "UserID",
        "AttributeType": "S"
      }
    ],
    "TableName": "Notifications",
    "KeySchema": [
      {
        "AttributeName": "UserID",
        "KeyType": "HASH"
      },
      {
        "AttributeName": "Timestamp",
        "KeyType": "RANGE"
      }
    ],
    "TableStatus": "CREATING",
    "CreationDateTime": "2025-01-01T16:52:18.064000+00:00",
    "ProvisionedThroughput": {
      "NumberOfDecreasesToday": 0,
      "ReadCapacityUnits": 5,
      "WriteCapacityUnits": 5
    },
    "TableSizeBytes": 0,
    "ItemCount": 0,
    "TableArn": "arn:aws:dynamodb:us-east-1:515966506771:table/Notifications",
    "TableId": "1e85731e-cc60-4fdf-9779-70a99d730790",
    "DeletionProtectionEnabled": false
  }
}

```

Microsoft Windows [Version 10.0.22631.4602]
 (c) Microsoft Corporation. All rights reserved.

```

C:\Users\Shubham>aws dynamodb list-tables --region us-east-1
{
  "TableNames": [
    "Events",
    "Notifications",
    "Tickets"
  ]
}

```

Step 3: Deploy Services

1. Set Up Local Development Environment:

Install dependencies for your chosen language/ framework:

For Python (FastAPI):

```
pip install fastapi uvicorn boto3
```

2. Develop Event Registration Service:

Endpoints:

- POST /events: Create a new event.
- GET /events: Fetch a list of events.
- POST /register: Register a user for an event.

Example (event-registration):

```
from fastapi import FastAPI
import boto3

app = FastAPI()
dynamodb = boto3.resource('dynamodb')
events_table = dynamodb.Table('Events')

@app.post("/events")
def create_event(event_id: str, details: dict):
    events_table.put_item(Item={"EventID": event_id, "EventDetails": details})
    return {"message": "Event created successfully"}
```

Similarly, we have created Backend services for ticketing and user engagement.

Step 4: Deploy Services

1. Containerize Services:

Write a Dockerfile for each service.

Example (event-registration):

```
event-registration > Dockerfile > ...
1  # Use a Node.js base image
2  FROM node:16
3
4  # Set the working directory
5  WORKDIR /usr/src/app
6
7  # Copy package.json and package-lock.json
8  COPY package*.json ./
9
10 # Install dependencies
11 RUN npm install
12
13 # Copy the rest of the application code
14 COPY . .
15
16 # Expose the service port
17 EXPOSE 3001
18
19 # Start the service
20 CMD ["npm", "start"]
21
```

Similarly, we have created Dockerfile for ticketing and user engagement.

2. Push to AWS ECR (Elastic Container Registry):

Create a repository for each service in ECR.

Build and push the Docker images.

- `aws ecr create-repository --repository-name event-service`
- `docker build -t event-service .`
- `docker tag event-service:latest <account_id>.dkr.ecr.<region>.amazonaws.com/event-service:latest`
- `docker push <account_id>.dkr.ecr.<region>.amazonaws.com/event-service:latest`

Created Cluster

```
PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management> aws --version
aws-cli/2.22.26 Python/3.12.6 Windows/11 exe/AMD64
PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management> aws ecs list-clusters
{
  "clusterArns": []
}
PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management> aws ecs create-cluster --cluster-name distributed-event-cluster
{
  "cluster": {
    "clusterArn": "arn:aws:ecs:us-east-1:515966506771:cluster/distributed-event-cluster",
    "clusterName": "distributed-event-cluster",
    "status": "ACTIVE",
    "registeredContainerInstancesCount": 0,
    "runningTasksCount": 0,
    "pendingTasksCount": 0,
    "activeServicesCount": 0,
    "statistics": [],
    "tags": [],
    "settings": [
      {
        "name": "containerInsights",
        "value": "disabled"
      }
    ],
    "capacityProviders": [],
    "defaultCapacityProviderStrategy": []
  }
}
```

Created Repos

```
PS C:\Users\Shubham> aws --version
aws-cli/2.22.26 Python/3.12.6 Windows/11 exe/AMD64
PS C:\Users\Shubham> aws ecr create-repository --repository-name event-registration-service
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:515966506771:repository/event-registration-service",
    "registryId": "515966506771",
    "repositoryName": "event-registration-service",
    "repositoryUri": "515966506771.dkr.ecr.us-east-1.amazonaws.com/event-registration-service",
    "createdAt": "2025-01-02T17:05:34.292000+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
PS C:\Users\Shubham> aws ecr create-repository --repository-name ticketing-service
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:515966506771:repository/ticketing-service",
    "registryId": "515966506771",
    "repositoryName": "ticketing-service",
    "repositoryUri": "515966506771.dkr.ecr.us-east-1.amazonaws.com/ticketing-service",
    "createdAt": "2025-01-02T17:05:37.427000+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

```
PS C:\Users\Shubham> aws ecr create-repository --repository-name user-engagement-service
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:515966506771:repository/user-engagement-service",
    "registryId": "515966506771",
    "repositoryName": "user-engagement-service",
    "repositoryUri": "515966506771.dkr.ecr.us-east-1.amazonaws.com/user-engagement-service",
    "createdAt": "2025-01-02T17:05:40.735000+00:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

Check AWS Login

```
PS C:\Users\Shubham> aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 515966506771.dkr.ecr.us-east-1.amazonaws.com
Login Succeeded
PS C:\Users\Shubham>
```

Built and Push the Docker images.

```
PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management> cd .\event-registration\
PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management\event-registration> docker build -t event-registration-service .
[*] Building 2.3s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 385B
=> [internal] load metadata for docker.io/library/node:16
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/node:16@sha256:f77a1aef2da8d83e45ec990f45df50f1a286c5fe8bbfb8c6e4246c6389785c0b
=> => resolve docker.io/library/node:16@sha256:f77a1aef2da8d83e45ec990f45df50f1a286c5fe8bbfb8c6e4246c6389785c0b
=> [internal] load build context
=> => transferring context: 141B
=> CACHED [2/5] WORKDIR /usr/src/app
=> CACHED [3/5] COPY package*.json ./
=> CACHED [4/5] RUN npm install
=> CACHED [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:f6ad7c77d86443f71cca36cf4e5c1fd7f36b452be2a9802224ce27297e352f71
=> => exporting config sha256:0fd5a5dd62ad48125e763485fcbdcac5c7b75a78f8551bb4e7835186e8248d6c
=> => exporting attestation manifest sha256:120b199a082a9cf091d9e4ab76ce3c6fe1e40194dc23a83904c4f1b663c5116d
=> => exporting manifest list sha256:c9c47e237ffcb2eafbb51227bc6ef60d9a818f664f9ebb4cae866796828aa23
=> => naming to docker.io/library/event-registration-service:latest
=> => unpacking to docker.io/library/event-registration-service:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/n5nazpop6ld4muybpsaa2nbyo

What's next:
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management\event-registration> docker push 515966506771.dkr.ecr.us-east-1.amazonaws.com/event-regi-
stration-service:latest
The push refers to repository [515966506771.dkr.ecr.us-east-1.amazonaws.com/event-registration-service]
tag does not exist: 515966506771.dkr.ecr.us-east-1.amazonaws.com/event-registration-service:latest

PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management\event-registration> docker push 515966506771.dkr.ecr.us-east-1.amazonaws.com/event-regi-
stration-service:latest
The push refers to repository [515966506771.dkr.ecr.us-east-1.amazonaws.com/event-registration-service]
b41984fdb36: Pushed
7ed9f114588c: Pushed
513d77925604: Pushed
311d6c465ea: Pushed
dbc6facec679: Pushed
ca266fd61921: Pushed
7863ea37eda: Pushed
ee7d78be1eb9: Pushed
ae3b95bbaa61: Pushed
32357571887c: Pushed
ff09397e94b7: Pushed
a9fcb3312e2: Pushed
8e421f66aff4: Pushed
latest: digest: sha256:c9c47e237ffcb2eafbb51227bc6ef60d9a818f664f9ebb4cae866796828aa23 size: 856
```

Similar to ticketing service.

```
PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management\ticketing> docker build -t ticketing-service .
[*] Building 2.3s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 383B
=> [internal] load metadata for docker.io/library/python:3.9
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/python:3.9@sha256:eeb155c46fdd3f03ff0e5efc8cac0c475b31eb4ea92c4b85231280b663a6f9a0
=> => resolve docker.io/library/python:3.9@sha256:eeb155c46fdd3f03ff0e5efc8cac0c475b31eb4ea92c4b85231280b663a6f9a0
=> [internal] load build context
=> => transferring context: 93B
=> CACHED [2/5] WORKDIR /usr/src/app
=> CACHED [3/5] COPY requirements.txt .
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt
=> CACHED [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:9ac0c1f93bb476602b78131001ab63078fa884723108de804c5258316ee278abf
=> => exporting config sha256:39ac6db2538dee4603fedc6e902cdeff3c195e68a6ab31a14c5e87fc0b2dc7d2
=> => exporting attestation manifest sha256:db735bb1c095d63e897e3dce4d072b3c41da553dc90c8c4e6b2115fd291804a7
=> => exporting manifest list sha256:55d70f955bc8ebf479712f892e2c382d3899b676d834b241eb1c12bdea60d79
=> => naming to docker.io/library/ticketing-service:latest
=> => unpacking to docker.io/library/ticketing-service:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/z5w4czgli56xfissg24w15e5d

What's next:
  View a summary of image vulnerabilities and recommendations → docker scout quickview

PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management\ticketing> docker tag ticketing-service:latest 515966506771.dkr.ecr.us-east-1.amazonaws.com/ticketing-service:latest
```

```

PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management\ticketing> docker push 515966506771.dkr.ecr.us-east-1.amazonaws.com/ticketing-service:latest
The push refers to repository [515966506771.dkr.ecr.us-east-1.amazonaws.com/ticketing-service]
e17c4285738c: Pushed
7f3e1d3dbcee: Pushed
f079c5474f57: Pushed
b6620d26bcfc: Pushed
0a96bdb82805: Pushed
1e41acfc3c57e: Pushed
54c7be425079: Pushed
67c94ce4a893: Pushed
80472c9b4e6a: Pushed
fa936c15d01f: Pushed
7aa8176e6d89: Pushed
1523f4b3f560: Pushed
latest: digest: sha256:55d70f955bc8ebf47971f2f892e2c382d3099b676d834b241eb1c12bdea60d79 size: 856

```

Similar to user-engagement service.

```

PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management\user-engagement> docker build -t user-engagement-service .
[+] Building 2.3s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 303B
=> [internal] load metadata for docker.io/library/python:3.9
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/python:3.9@sha256:eeb155c46fdd3f03ff0e5efc8cac8c475b31eb4ea92c4b85231280b663a6f9a0
=> => resolve docker.io/library/python:3.9@sha256:eeb155c46fdd3f03ff0e5efc8cac8c475b31eb4ea92c4b85231280b663a6f9a0
=> [internal] load build context
=> => transferring context: 93B
=> CACHED [2/5] WORKDIR /usr/src/app
=> CACHED [3/5] COPY requirements.txt
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt
=> CACHED [5/5] COPY .
=> => exporting layers
=> => exporting manifest sha256:48caea7c1ac6ad774143be8faa7c9eeab3494795468a802238bd085ff9bcd322b
=> => exporting config sha256:81e8c92f9918922a147ea3fc307b2b89538aaf375028ad79df0f48e785b5be58
=> => exporting attestation manifest sha256:da815c62e71334108f6c315a02c08a46d0384ff4dc7e6970b3613f3c1ddbb0
=> => exporting manifest list sha256:2a5334063bf3e6ef124cfc8c8b1f5804b5cfdad67d7fc6559ada2b541a8b86f
=> => naming to docker.io/library/user-engagement-service:latest
=> => unpacking to docker.io/library/user-engagement-service:latest

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/514vtg7h8ny080pvtm8alk14

What's next:
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS D:\UCD\COMP41720 Distributed Systems\distributed-event-management\user-engagement> docker tag user-engagement-service:latest 515966506771.dkr.ecr.us-east-1.amazonaws.com/user-engagement-service:latest
The push refers to repository [515966506771.dkr.ecr.us-east-1.amazonaws.com/user-engagement-service]
1523f4b3f560: Pushed
7aa8176e6d89: Pushed
f079c5474f57: Pushed
2062bd892cbl: Pushed
59d7a59e6e0: Pushed
0a96bdb82805: Pushed
54c7be425079: Pushed
c63f046c2bf1: Pushed
80472c9b4e6a: Pushed
fa936c15d01f: Pushed
b6620d26bcfc: Pushed
fa24cab1fb1b: Pushed
latest: digest: sha256:2a5334063bf3e6ef124cfc8c8b1f5804b5cfdad67d7fc6559ada2b541a8b86f size: 856

```

Deploying Backend locally:

Commands used : - docker compose build

- docker compose up

The screenshot displays a development environment in VS Code. The Explorer sidebar on the left shows a project structure for 'distributed-event-management'. The main editor area shows the 'app.py' file with a function 'add_ticket()' that checks if a field is present in a dictionary and returns a JSON response. The Output window at the bottom shows the Docker Compose build process, and the Terminal window shows the Docker Compose up command being executed, resulting in the creation of several containers and their startup logs.

Testing on Postman:

http://localhost:3001/events

GEThttp://localhost:3001/events

Send

ParamsAuthorizationHeaders (6)BodyScriptsSettings

Query Params

Key	Value	Description
-----	-------	-------------

BodyCookiesHeaders (6)Test Results

200 OK118 ms1.4 KB

PrettyRawPreviewVisualizeJSON

```
1 [
2   {
3     "EventDetails": {
4       "Date": "2024-12-15T10:00:00Z",
5       "Description": "Annual conference for technology enthusiasts.",
6       "Location": "New York, NY",
7       "Name": "Tech Conference 2025"
8     },
9     "EventID": "001",
10    "MaxCapacity": "500",
11    "OrganizerID": "123",
12    "RegisteredUsers": [
13      "\u0001", "\u0002", "\u0003"
14    ]
15  },
16  {
17    "EventDetails": {
18      "date": "2025-05-20",
19      "name": "Google Conference 2025"
20    },
21    "EventID": "event234",
22    "MaxCapacity": "70",
23    "OrganizerID": "organizer999",
24    "RegisteredUsers": "0"
25  }
26 ]
```

http://localhost:3002/

GEThttp://localhost:3002/

Send

ParamsAuthorizationHeaders (6)BodyScriptsSettings

HeadersHide auto-generated headers

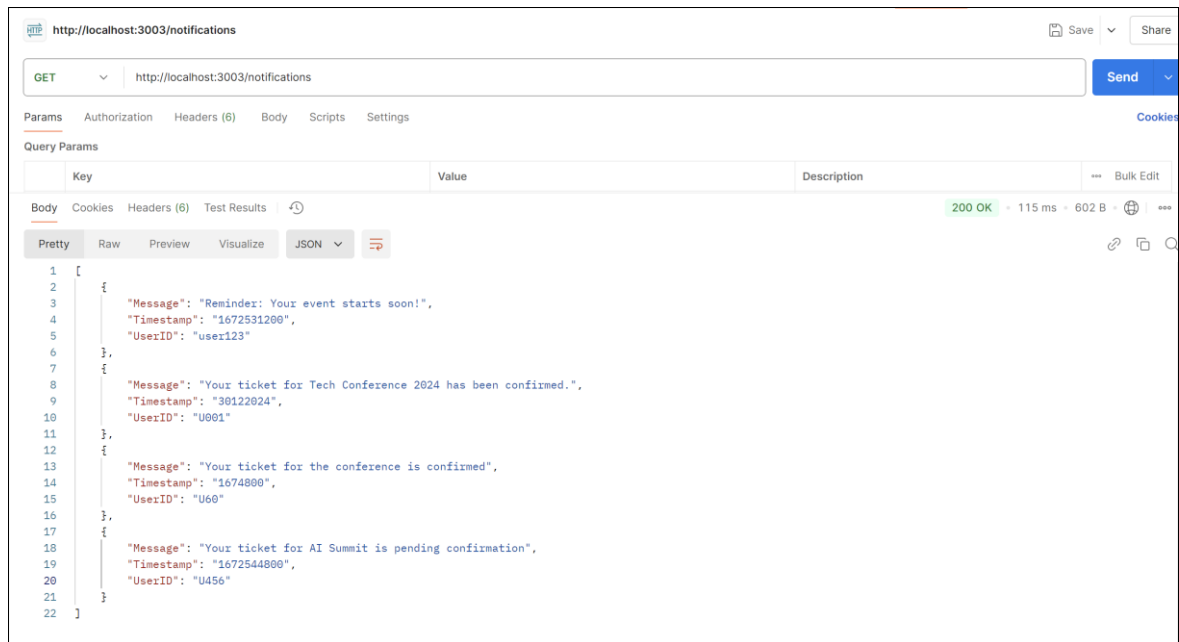
Key	Value	Description
Postman-Token	<calculated when request is sent>	
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.43.0	

BodyCookiesHeaders (6)Test Results

200 OK16 ms241 B

PrettyRawPreviewVisualizeJSON

```
1 {
2   "message": "Ticketing Service is running!"
3 }
```

For Front-end:

Step 1: Set up the React application

1.1. Install Node.js

- Download and install Node.js from the official website
- Verify the installation:
`node -v`
`npm -v`

1.2. Create a React Project

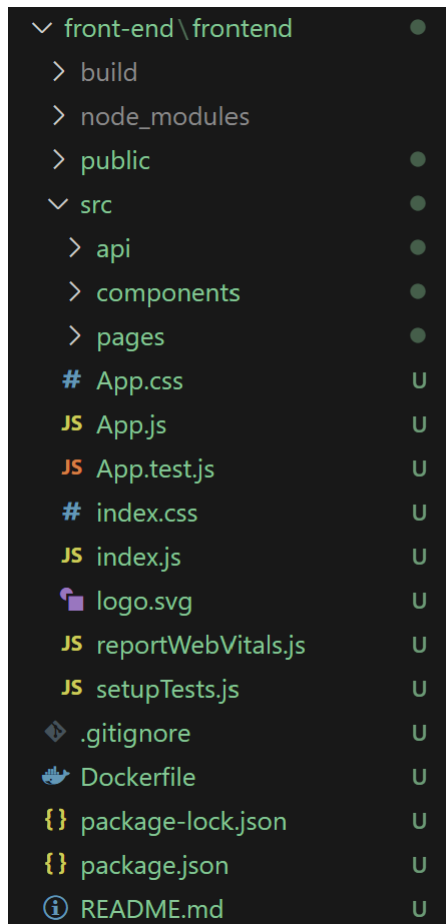
- Run the following command to create a new React app:
`npx create-react-app frontend`
- Navigating to the project directory:
`cd frontend`
- Starting the development server:
`npm start`
The React app will open in default browser: <http://localhost:3000>

Step 2: Build the Frontend

2.1. Installing required libraries:

- `npm install axios react-router-dom bootstrap socket.io-client`

2.2. Setting the Project structure:



2.3. Creating the .js files in components, api and pages folder

Step 3: Test Locally

3.1. Starting backend services using Docker Compose:

- docker compose build
- docker compose up

3.2. Starting the React development server using the command:

- npm start

3.3. Open “http://localhost:3000” in your browser to interact with the frontend

Step 4: Build and Deploy the Frontend

4.1. Build the React app:

- npm run build : This creates an optimized production build in the build directory.

4.2. Deploy to AWS S3 and CloudFront

- Create an S3 Bucket
 - Used the AWS Management Console.
 - Enable static website hosting for the bucket.
- Upload files to S3
 - Upload the contents of the build directory to the S3 bucket

- Set up CloudFront
 - Configure an AWS CloudFront distribution for the S3 bucket to improve delivery performance and enable HTTPS.

Step 5: Final Testing

Access the deployed frontend from the CloudFront distribution.

The screenshot displays the AWS CloudFront console interface. The left-hand navigation pane shows the 'CloudFront' service selected, with the 'Distributions' tab active. The main content area is titled 'E1NMAGVQEJ9756' and includes a 'View metrics' button. Below the title, there are tabs for 'General', 'Security', 'Origins', 'Behaviors', 'Error pages', 'Invalidations', 'Tags', and 'Logging'. The 'General' tab is currently selected, showing two main sections: 'Details' and 'Settings'.

Details Section:

- Distribution domain name:** d6ld0sn4eag3v.cloudfront.net
- ARN:** arn:aws:cloudfront::515966506771:distribution/E1NMAGVQEJ9756
- Last modified:** January 4, 2025 at 10:37:57 AM UTC

Settings Section:

- Description:** -
- Alternate domain names:** -
- Standard logging:** Off
- Cookie logging:** Off
- Default root object:** index.html
- Price class:** Use North America, Europe, Asia, Middle East, and Africa
- Supported HTTP versions:** HTTP/2, HTTP/1.1, HTTP/1.0

An 'Edit' button is located in the top right corner of the Settings section.