DISASTER RECOVERY PROJECT ON IBM VIRTUAL CLOUD SERVER

OVERVIEW:

This documentation provides an overview of the Disaster Recovery project, which involves deploying a To-Do List web application on IBM Cloud Virtual Servers. The project includes server-side code development and a disaster recovery plan.

SERVER SCRIPT:

Description:

The server script is responsible for handling HTTP requests, serving static files, and managing To-Do list data.

File: server.js

Dependencies:

Express.js: A web application framework for Node.js.

body-parser: Middleware for parsing request bodies.

url: Node.js module for working with URLs.

path: Node.js module for working with file and directory paths.

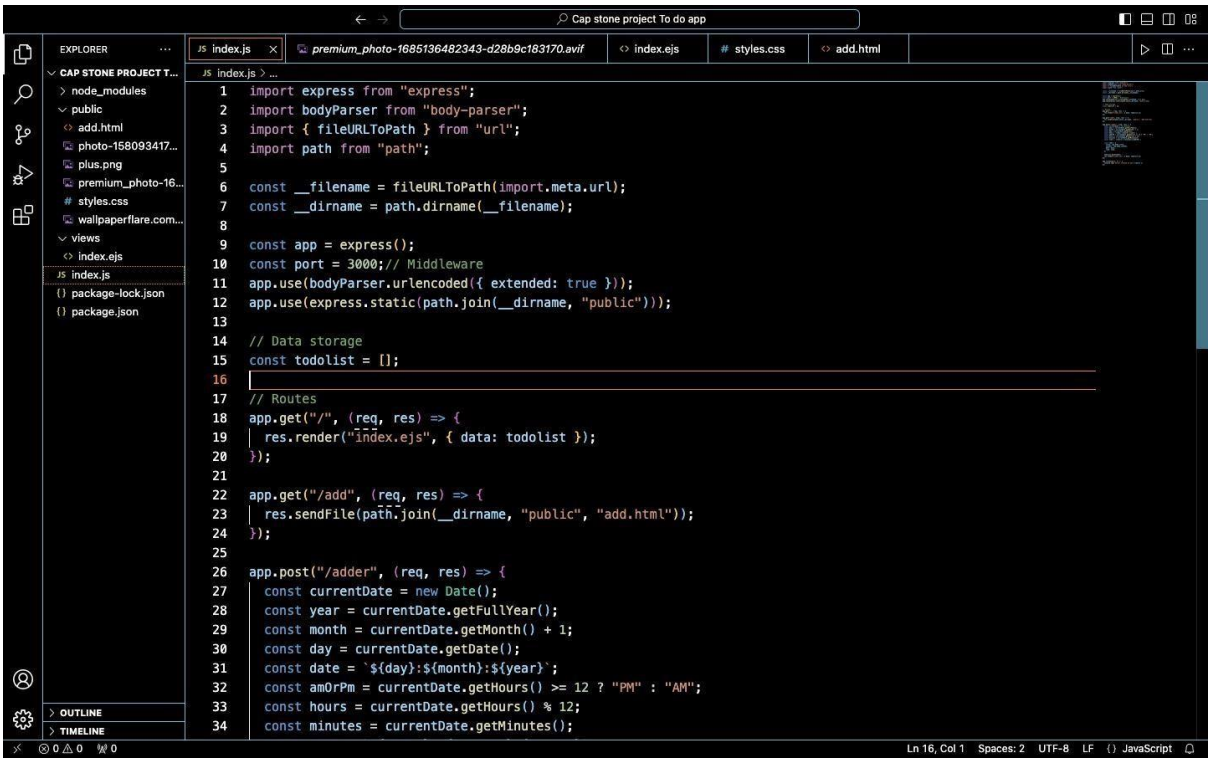
Initialization:

The server is initialized using Express.js on a specified port (3000).

Middleware:

The server uses the body-parser middleware for parsing form data in POST requests.

Static files (HTML, CSS, JavaScript) are served using express.static middleware from the "public" directory.



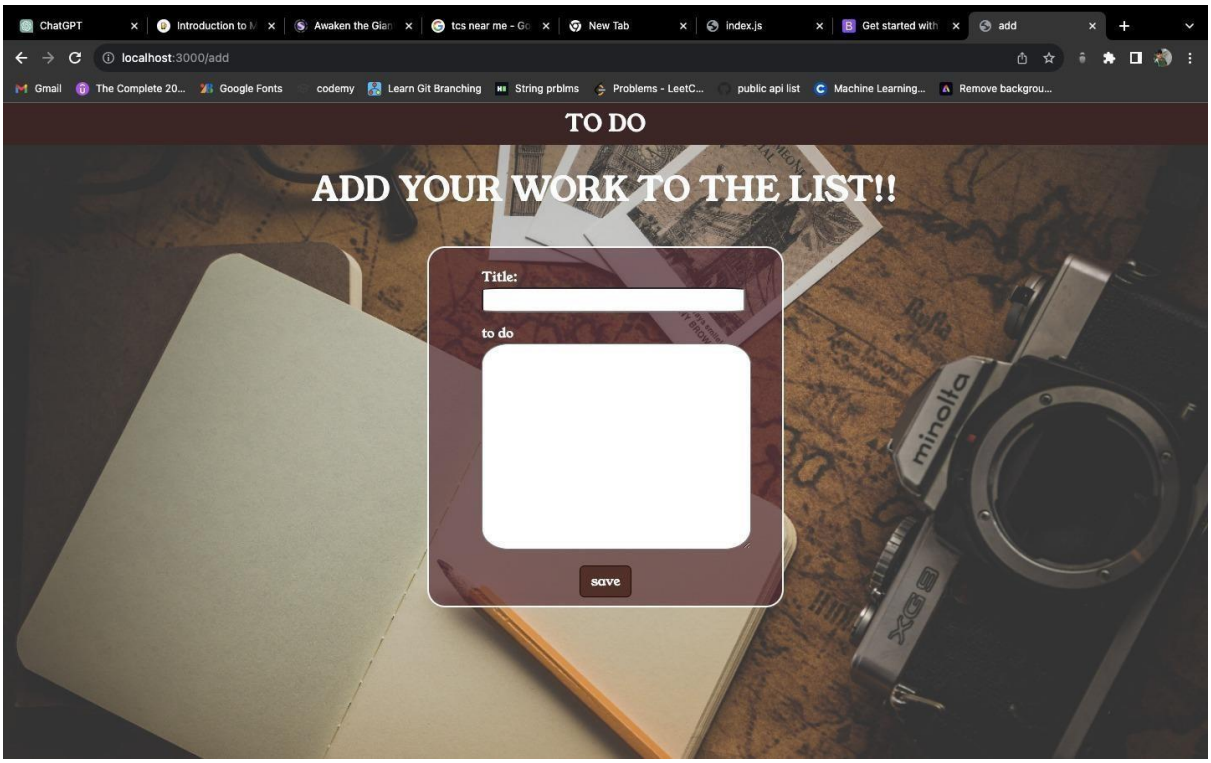
Data Storage:

To-Do list data is currently stored in memory as an array named todolist.

Routes:

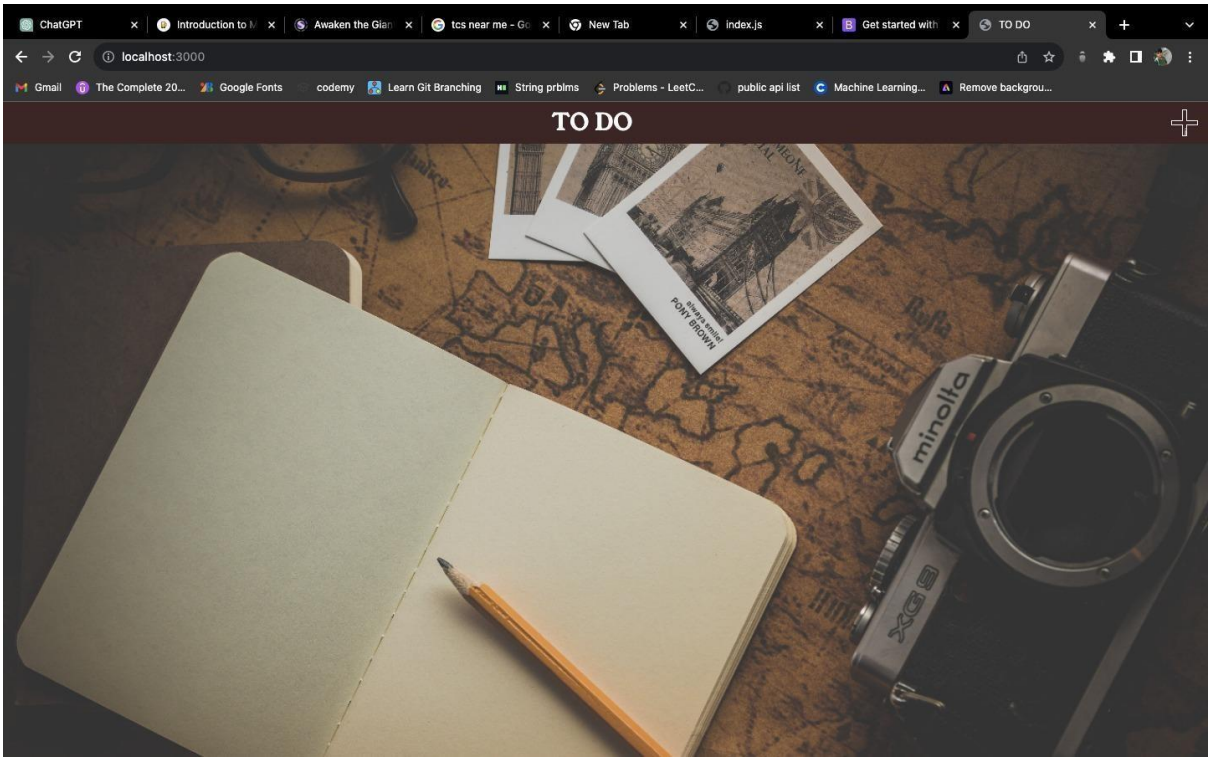
GET /: Renders the To-Do list on the homepage.

GET /add: Serves the HTML form for adding new tasks.

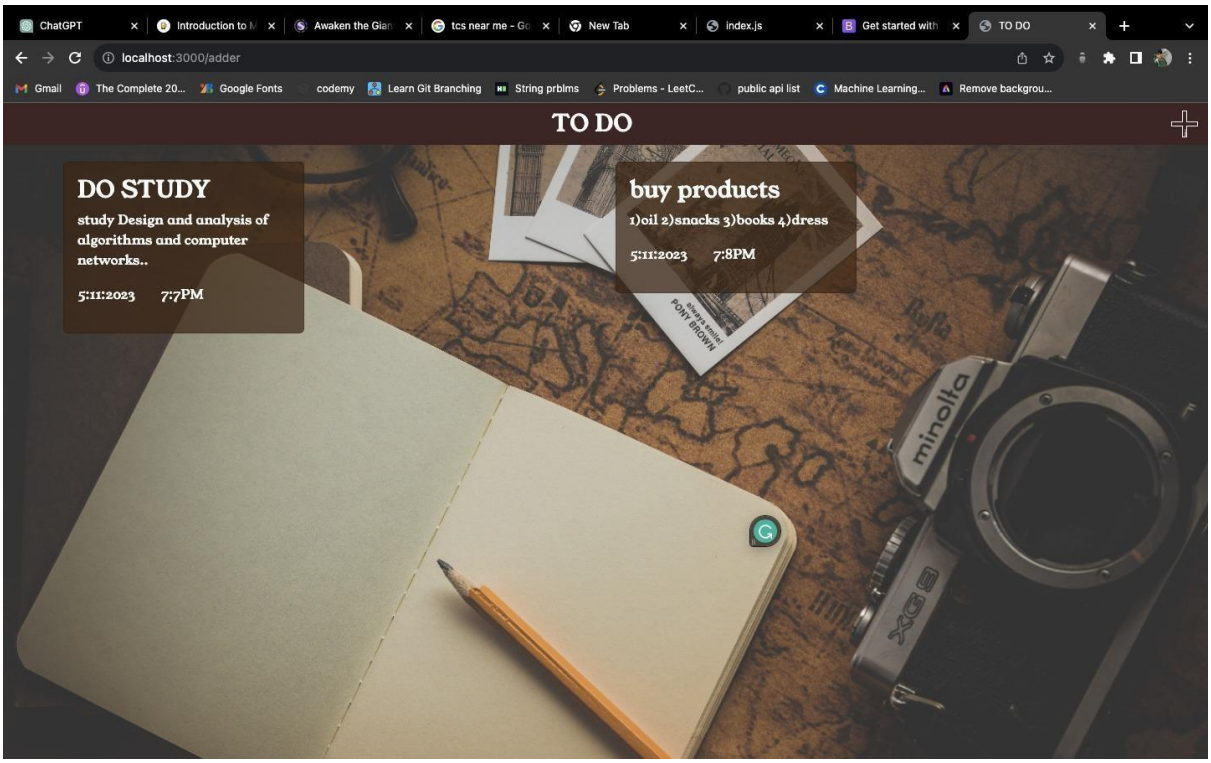


POST /adder: Processes the form submission, adds a new task to the To-Do list, and re-renders the To-Do list.

BEFORE ADDING THE TASK:

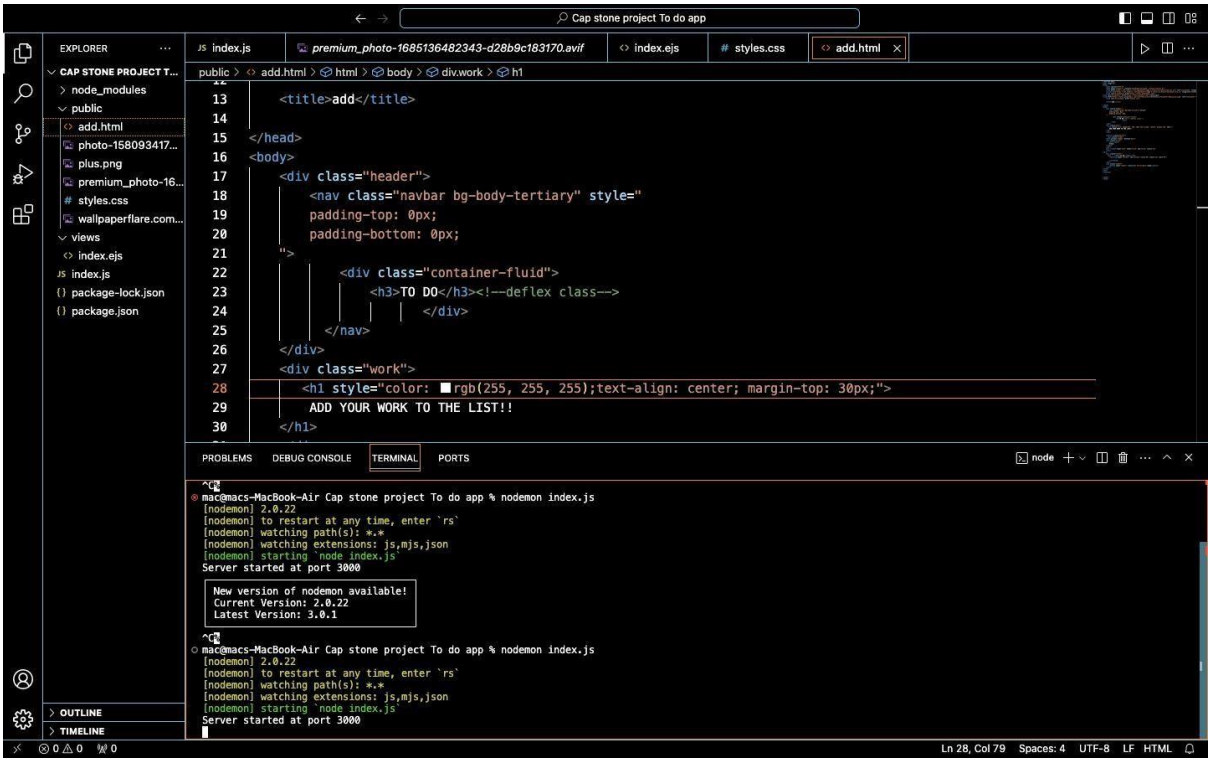


RENDERING THE TO DO LIST AFTER PROCESSING THE FORM SUBMISSION:



Date and Time Handling:

The server script captures the current date and time for each added task.



Starting the Server:

The server is started with a message indicating the port it's running on.

Cloud Database Connection:

Description:

The To-Do List application integrates with a cloud database for persistent data storage. MongoDB is used for this purpose.

Connection Configuration:

1.The server script connects to the MongoDB cloud database using the official MongoDB Node.js driver.

Screenshots Of the Configuration Settings for Connecting to the Cloud Database:

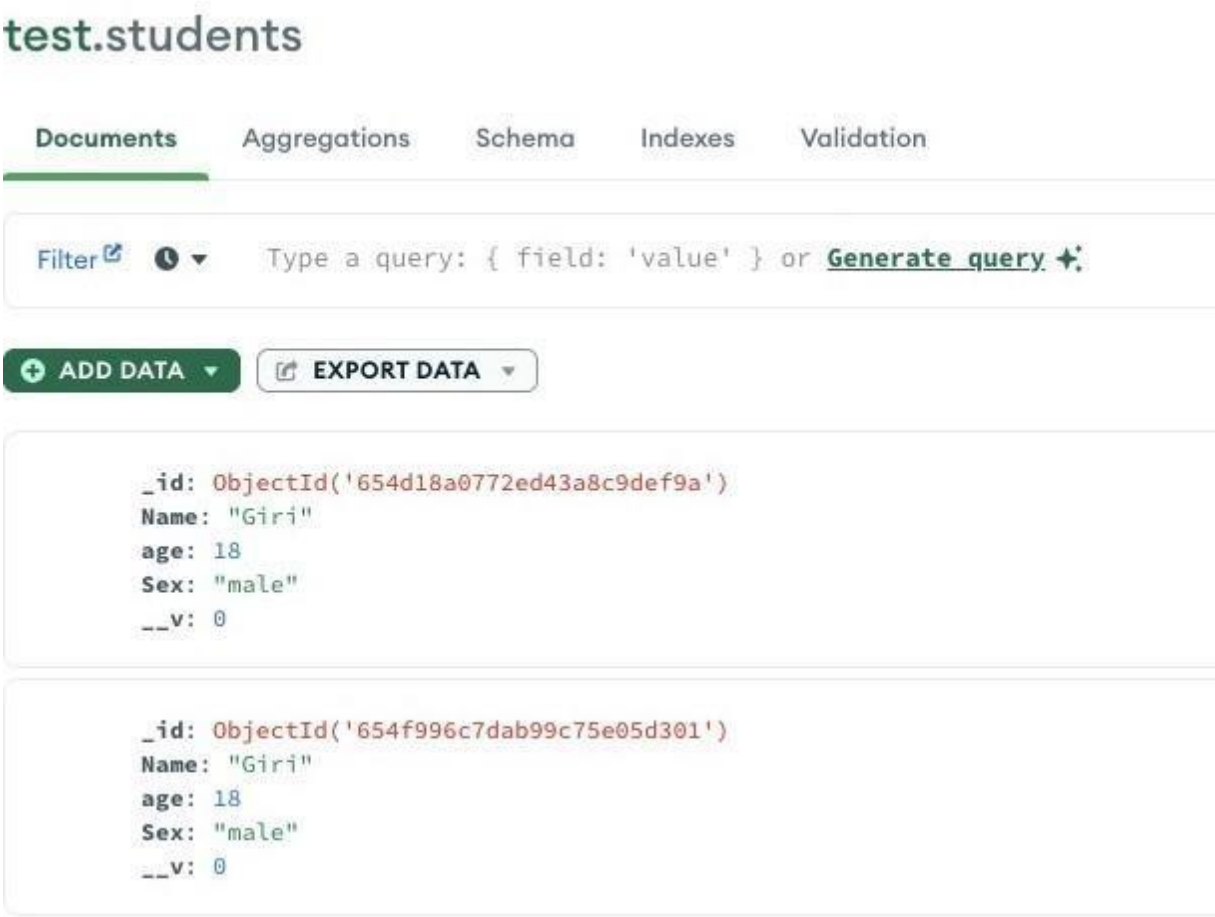
```
app.get("/add",(req,res)=>
{
   const stud1=new student({Name:"Giri",age:18,Sex:"male"});
   stud1.save();
    res.send("succesfully added");
});
app.get("/retr",async(req,res)=>{
    const data =await student.find({}).exec();
    console.log(data);
    res.json(data);

  });
app.get("/update",async(req,res)=>{
const resu=await student.updateOne({_id:"654d01a3a5a865b45ec9e87b"},{age:19});
if(resu.acknowledged)
    res.send("successfullu");
});

app.get("/delete",async(req,res)=>{
    const cnt=await student.deleteOne({_id:"654d0ea8ed4ef06dfa84852a"});

    res.send("dele successfully");
})
```

Schema of the database collection:

```
const dataschema=new mongoose.Schema({
    Name:String,
    age:Number,
    Sex:String,
});
const student=mongoose.model("student",dataschema);
```

Adding the data to the cloud database:

```
_id: ObjectId('654d18a0772ed43a8c9def9a')
Name: "Giri"
age: 18
Sex: "male"
__v: 0
```



Database Schema:

The code snippet regarding the schema of the database collection is attached above.

Usage of Cloud Database:

The data is retrieved and stored in the cloud database. The code snippets related to this are attached above.

Disaster Recovery Plan:

Description:

The disaster recovery plan outlines measures for ensuring the high availability and reliability of the To-Do List application on IBM Cloud Virtual Servers.

Components:

1.**Backup and Recovery:**

Backup Strategy: We implement a daily backup strategy to ensure the safety of critical data and configurations. Backups include both incremental and full backups, stored securely in an offsite location.

Data Recovery Procedures: In the event of data loss or corruption, our recovery procedures involve restoring the latest validated backup. Detailed step-by-step instructions are available to facilitate a quick and efficient recovery process.

Backup Validation: Regular checks are performed to validate the integrity and completeness of backups. This ensures that backups are reliable and can be used for effective data recovery.

2.**Failover and Load Balancing:**

Failover Mechanisms: To ensure high availability, we have implemented failover mechanisms that automatically redirect traffic in case of server failures. This minimizes downtime and ensures uninterrupted service.

Load Balancing Configuration: Our load balancing configuration evenly distributes incoming traffic across multiple servers. This not only optimizes server utilization but also provides redundancy in case of server failures.

3.**Monitoring and Alerts:**

Health Monitoring: Continuous health monitoring systems are in place to assess the status of servers, databases, and network connections. Any anomalies or performance degradation triggers alerts for immediate attention.

Alerting Systems: Real-time alerts are configured to notify the operations team of any abnormal conditions or failures. These alerts are delivered through multiple channels, ensuring timely awareness and response.

4.**Testing and Validation**:

Recovery Testing: Regular disaster recovery tests are conducted to validate the effectiveness of our recovery procedures. These tests simulate various disaster scenarios to ensure our team is well-prepared for any situation.

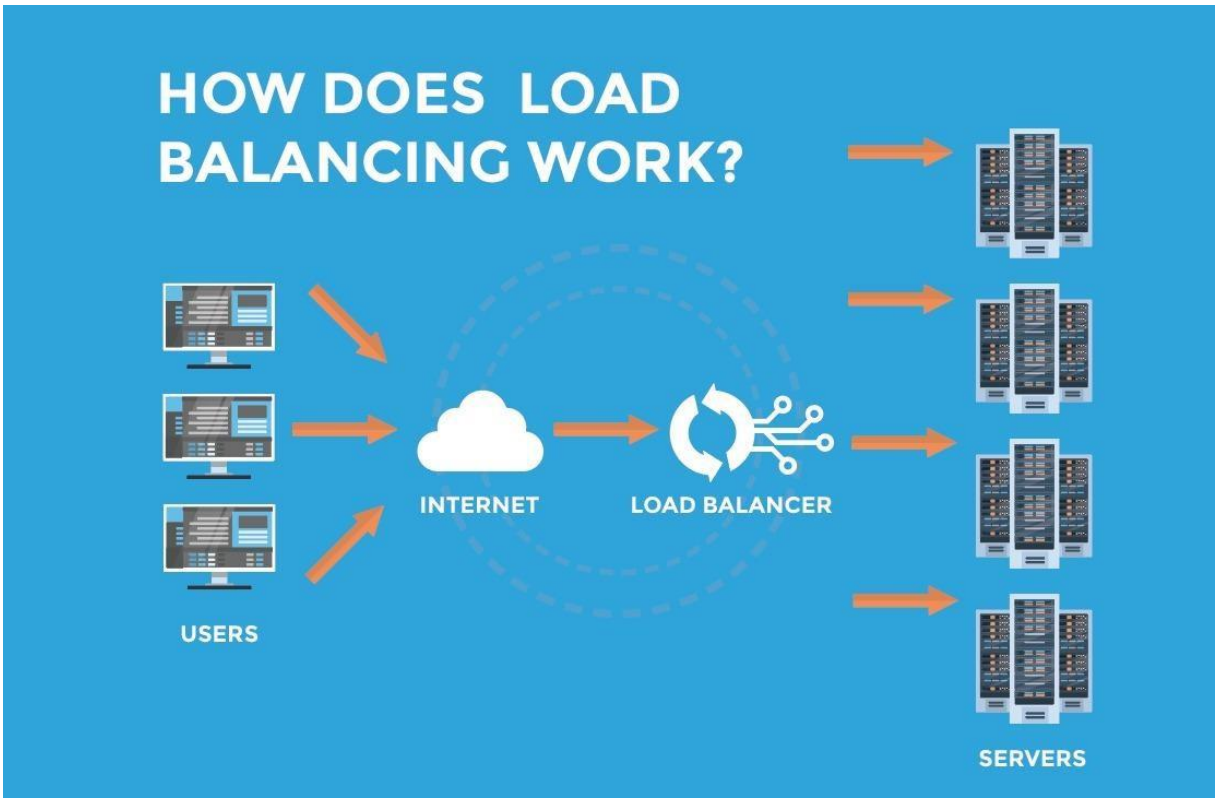
Documentation Validation: The accuracy and completeness of our disaster recovery documentation are regularly validated. Any updates or changes are promptly reflected in the documentation to ensure it remains a reliable reference.

5**.Documentation Updates**:

Change Control Procedures: Changes to the disaster recovery plan are managed through a structured change control process. This includes versioning and clear documentation of any modifications made.

Review Schedule: A schedule is established for periodic reviews of the disaster recovery plan. This ensures that the plan is up-to-date with any changes in the system architecture or business requirements.

Screenshots of Disaster Recovery Setup:



SEVERITY	TOTAL ALERTS	NEW	ACKNOWLEDGED	CLOSED
Sev 0	3138	3138	0	0
Sev 1	13656	13078	0	578
Sev 2	1724	1724	0	0
Sev 3	10320	9883	2	435
Sev 4	5600	5307	0	293

Future Development:

The current project is at a development stage and does not yet include features such as user authentication or database integration. Future development steps may include:

1.Integrating a MongoDB database for persistent data storage.

2.Implementing user authentication to allow multiple users to manage their own tasks.

3.Enhancing the user interface and adding features like task editing and completion status.

4.Ensuring security measures to protect user data and application resources.

Conclusion:

This documentation provides an initial overview of the To-Do list application's server script and disaster recovery plan. Future updates and enhancements will be made to expand the functionality of the application and the disaster recovery capabilities.