SOFTWARE DESIGN AND ARCHITECHURE

ASSIGNMENT

NAME: ABDUL GHAFFAR ***REG NO: FA22-BSE-021***

JANUARY 4, 2025

Software Design and Architecture Assignment

Part 1 :

Five Common Architectural Problems

Problem 1: Scalability Challenges

Issue:

Websites often face slowness or downtime when user traffic increases, especially if the system wasn't built to handle growth.

Example:

A blog designed for a small audience struggles when thousands of users visit at the same time, leading to crashes.

Fix:

Add more servers using load balancers to share the traffic or use cloud-based solutions for scalability.

Problem 2: Inefficient Database Setup

Issue:

Without proper indexing or optimization, database searches become slow as the data grows.

Example:

Searching for items on an e-commerce site takes several seconds because the database isn't optimized.

Fix:

Use indexing on frequently searched fields and consider caching mechanisms for faster data retrieval.

Problem 3: Unorganized Code Structure

Issue:

Code duplication and tightly coupled components make it difficult to manage or update the system.

Example:

Payment functionality is coded separately in multiple files, creating inconsistency and more maintenance work.

Fix:

Break down the code into reusable modules and follow best practices like DRY (Don't Repeat Yourself).

Problem 4: Unbalanced Server Load

Issue:

Some servers get overwhelmed with traffic while others remain underutilized, leading to inefficiency.

Example:

A social platform routes all user requests to one server, causing delays and crashes.

Fix:

Distribute traffic evenly using algorithms like round-robin or based on server capacity.

Problem 5: Security Gaps

Issue:

Weak login systems or improper input validation can make a system vulnerable to attacks.

Example:

Attackers exploit a website that doesn't sanitize input fields, causing security breaches.

Fix:

Implement strong authentication methods and validate all user inputs to prevent such attacks.

Part 2:

<u>Demonstration of a Problem and Solution</u>

- **Issue Demonstrated: Slow Database Queries**

Scenario:

A simple product search on a Node.js website is sluggish due to an unoptimized database.

- **Steps to Recreate the Problem:**

1. Create a database **MSSQL (AS I AM EXPERIENCED IN MSSQL)** named **"architecture"** with a **"products"** collection.

2. Add some product data without setting up any indexes.

3. Use a **Node.js Express** server **(USING IT SO FAR)** to create a search feature.

4. Notice how long it takes to search for products.

- **Solution Steps:**

1. Add an index to the **"name"** field in the MSSQL collection.

2. Re-test the search functionality to see improved speed.
(**Retesting will allow us to see the speed and efficiency difference in the architecture)**

Node.js Code Example

```
 const express = require('express');
const sql = require('sqlexpress//');

const app = express();
const PORT = 3000;
```

// connecting to mssql express server

```
mssql. Connect('mssql://localhost:27017/architecture', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  });

  const productSchema = new mssql.Schema({
    name: String,
    price: Number,
  });

  const Product = mssql.model('Product', productSchema);

  // Seed database
  app.get('/seed', async (req, res) => {
    await Product.insertMany([
      { name: 'Laptop', price: 1000 },
      { name: 'Phone', price: 500 },
      { name: 'Tablet', price: 300 },
    ]);
    res.send('Database seeded');
  });

  // Search endpoint
  app.get('/search', async (req, res) => {
    const query = req.query.name;
    const results = await Product.find({ name: { $regex: query, $options: 'i' }
});
    res.json(results);
  });

  // Add index (manual step to improve performance)
  app.get('/add-index', async (req, res) => {
    await Product.collection.createIndex({ name: 1 });
    res.send('Index added');
  });

  app.listen(PORT, () => console.log(`Server running on
http://localhost:${PORT}`));
```