**21BRS1161 Nishchhal Raturi**

**Review two Presentation**

1. **Domain model**
2. **Interaction diagrams**
3. **Class diagram and interface specification**
4. **System Architecture and System Design**

**1) Domain Model:**

**i) Concept Definitions:**

|  |  |  |
| --- | --- | --- |
| **Responsibility Description** | **Type** | **Concept** |
| R01: Conducts port scanning operations on target IP addresses | D | Scanner |
| R02: Stores the results of port scanning operations, including open ports and services. | K | ScanResult |
| R03: Stores scan results in a structured format for future retrieval and analysis | D | Database |
| R04: Provides a web interface for interacting with the port scanner, including triggering scans and retrieving results. | D | API |
| R05: Initiates port scanning operations and retrieves scan results through the API interface | D | User |

**ii) Association Definitions:**

|  |  |  |
| --- | --- | --- |
| **Concept Fair** | **Association Description** | **Association Name** |
| Scanner ↔ Database | Scanner stores scan results in the database | Stores |
| API ↔ Scanner | API triggers port scanning operations through the scanner | Initiates |
| User ↔ API | User interacts with the port scanner through the API interface | Uses |

**iii) Attribute Definitions:**

|  |  |  |
| --- | --- | --- |
| **Concept** | **Attribute** | **Attribute Definition** |
| Scanner | |  | | --- | | IP Address | | Port Range | | Scan Flags | | |  | | --- | | IP address of the target system to be scanned | | Range of ports to be scanned (e.g., 1-1024) | | Flags or options to be passed to the scanner for conducting the scan (e.g., "-sS" for SYN scan). | |
| ScanResult | |  | | --- | | IP Address | | Open Ports | | Services | | |  | | --- | | IP address of the target system to be scanned | | List of open ports discovered during the scan | | List of services running on the open ports. | |
| Database | |  | | --- | | Connection String | | Tables | | |  | | --- | | URL or connection parameters required to establish a connection to the database. | | Database tables used for storing scan results | |
| API | |  | | --- | | Endpoints | | Request Payload | | |  | | --- | | List of API endpoints exposed for triggering scans and retrieving results | | Data format expected by the API endpoints for triggering scans and providing input parameters | |

**iii)Traceability Matrix:**

| **Use Case** | **Project Work** | **Scanner** | **Scan Result** | **Database** | **API** | **User** |
| --- | --- | --- | --- | --- | --- | --- |
| UC1 | 9 | x |  | x | x | x |
| UC2 | 5 | x | x | x | x | x |
| UC3 | 8 | x |  | x | x | x |

**2)Interaction Diagrams**

**A diagram of a fast api

Description automatically generated**

**3.) Class diagram and interface specification**

**A diagram of a computer

Description automatically generated**

**Class Descriptions:**

**Scanner**: Represents the port scanner functionality of the system. It is responsible for scanning ports on the specified IP address using the given port range and scan flags.

Attributes:

ipAddress: The IP address to be scanned.

portRange: The range of ports to scan.

scanFlags: Flags or options used for scanning ports**.**

**ScanResult:** Represents the result of a port scan. It stores information about the open ports and the services running on them.

Attributes:

ipAddress: The IP address for which the scan result is generated.

openPorts: List of open ports found during the scan.

services: List of services running on the open ports.

**Database:** Manages the storage of scan results. It provides methods to connect to the database and store scan results.

Attributes:

connectionString: The connection string used to establish a connection with the database.

tables: Information about tables in the database where scan results are stored.

**API:** Represents the interface through which users interact with the system. It exposes endpoints for initiating port scans and retrieving scan results.

**Relationships:**

Scanner performs 1 to many scans (1..\*) resulting in ScanResult objects.

Database interacts with ScanResult to store the scan results.

API serves as an interface for users to interact with the system, including triggering port scans and retrieving scan results.

Scanner interacts with Database to store scan results.

API interacts with Database to fetch scan results and serve them to users.

**4) System Architecture and System Design:**

**Architectural Styles:**

Our design architecture is based on the client-server model, specifically a 2-tier architecture. In this model, clients directly connect with a centralized server, which acts as both a database and a service provider. All communication and data processing occur through the server, ensuring a unified infrastructure and centralized management of resources.

**Identifying Subsystems:**

Scanner: Conducts port scanning operations and interacts with the API for initiating scans.

Scan Result: Stores scan results in the database for future retrieval.

Database: Stores information related to scan results and system configuration.

API: Facilitates communication between clients and the server for triggering scans and retrieving results.

User Interface: Provides an interface for users to interact with the system, including initiating scans and viewing results.

**Mapping Subsystems to Hardware:**

The server runs on a master computer, hosting the database and serving as the central hub for data processing.

Clients, such as desktop computers or laptops, access the port scanner application through their web browsers or dedicated client software.

**Persistent Data Storage:**

The system stores scan results and configuration data in a relational database, ensuring long-term reliability and ease of data management.

Backups are scheduled to maintain data integrity and prevent data loss.

**Network Protocol:**

The system utilizes Microsoft Azure Mobile Services SQL database, offering scalability and flexibility in data management.

Data is stored in a cloud-based server, enabling seamless access and scalability based on demand.

**Global Control Flow:**

The system operates in a linear fashion, with each action dependent on the previous one.

Events, such as low inventory alerts, trigger responses in real-time to ensure efficient inventory management.

Concurrency is implemented through multi-threading, allowing multiple clients to initiate scans simultaneously and ensuring smooth system operation.