Name: Bhuwanesh Jung Thapa.
Class: B Tech 7th Sem CSE
Enroll: 40516403222.

Q.1) How does Input Validation prevent Injection Attacks on Cross-site Scripting (XSS)?

> Input validation is the process of checking and sanitizing all data entered by users before using it in the System. It is one of the most important techniques to prevent Injection attacks (SQL Injection, Command Injection) and XSS (cross site scripting)

How it prevents Injection Attacks.

- Injection attacks occur when an attacker injects malicious commands/code into an input field.

Example:

```
' OR = 1=1 --
```

Input Validation prevents this by:

1) Allowing only expected characters.
   - Example: username must only contain letters/numbers
2) Rejecting or sanitizing special characters (', ", ;, {}, <>)
3) Using prepared statements / parameterized queries.
   - so input is treated as data not executable code.
4) Ensuring correct data format.
   - Example: email must follow abc@gmail.com.
5) Blocking HTML tags / Script tags
6) Escaping Special Characters
   - convert < to < , > to >
7) Using whitelist Validation
   - only allow safe characters.
8) Encoding output before displaying on the page

Example:

user enters

```
<Script> alert ('xss') </Script>
```

(Q.2) Role of RESTful API in Integrating Frontend & Backend + Example Scenario.

- A RESTful API act as a bridge between the frontend (React, Vue, Angular) and backend (Node.js, Express, Django, etc). It allows communication using HTTP methods:
  - GET
  - POST
  - PUT
  - DELETE

The role of RESTful APIs in full stack development.

1) Enables communication between client & server.
2) Frontend sends request to API ; backend returns JSON data.
3) Provides structured endpoints
   Example:
   - api/products
   - api/users/login.
4) Allow seperation of concerns.
   - Frontend focuses on UI
   - Backend handles logic, authentication, databases.
5) Improves scalability & reusability
6) Standard format : JSON making integrations easy.

Example Scenario : fetching products in an online store.

Client (frontend React)

```
fetch ('https://api.example.com/products')
    .then (res => res.json())
    .then (data => console.log(data));
```

REST API (Backend Express.js)

```
app.get ('/products', (req, res) => {
    const products = [
      {id: 1, name: "Laptop", price: 50000},
      {id: 2, name: "phone", price: 20000}
    ];
    res.json(products);
});
```

(Q.3) Concept of Database Design + Normalization + Example (Online store)

- Database design means organizing data into tables, defining relationships, and ensuring data is:
  - Consistent
  - Scalable
  - free from redundancy
  - easy to query

Database designing includes:

1) Identifying entities (e.g. Users, products, orders)
2) Creating table structures
3) Defining primary keys, foreign keys
4) Ensuring relationship (1-to-1, 1-to many, many to many)
5) Applying Normalization to avoid redundancy.

Importance of Normalization

Normalization is the process of organizing tables to:

- Remove duplicate data
- Reduce update/delete anomalies
- Improve data integrity.

Levels include:

1NF: No repeating groups, atomic values
2NF: Remove partial dependency
3NF: Remove transitive dependency.

Example: online store Database

Entities:

- Customers
- Products
- Orders
- OrderItems.

Step1: Customer Table

customer_id (PK)	name	email

Step2: Product Table

product_id (PK)	product_name	price

Step 3: Order Table

order_id (PK) customer_id (FK) order_date.

Step 4: OrderItems Table

order_item_id (PK) order_id (FK) product_id (FK)

Quantity.

Q.4) Role of SQL Queries & ORM in Database Interactions for Web Applications.

SQL Queries

→ SQL (Structured Query language) allows applications to interact directly with relational databases.

SQL is used for

1) CRUD operations.
   - Create → INSERT
   - Read → SELECT
   - Update → UPDATE
   - Delete → DELETE

2) filtering data (WHERE)
3) Joining tables
4) Aggregations (COUNT, SUM, AVG)
5) Managing permissions, indexes, constraints.

Example SQL Queries.

```
SELCT * FROM products WHERE price < 20000;
INSERT INTO users (name, email) VALUES
    ("John", "john@gmail.com");
```

- ORM

Object Relational Mapping. allows developers to interact with the database using objects and classes instead of writing SQL manually.

Popular ORMs:

- Sequalize (Node.js)
- Prisma (Node.js)
- Hibernate (Java)
- Django ORM
- Entity Framework (.NET)

Example:

```
const User = sequelize.define("User", {
    name: Sequelize.STRING,
    email: Sequelize.STRING,
});

// Insert
User.create({
    name: "John",
    email: "john@gmail.com"});

// Fetch
User.findAll();
```