Backend Development with NodeJS

AMIT DHOJU

Collaboration With :

WorldLink
CORPORATE SOCIAL RESPONSIBILITY

THE ALGORITHM
Tech for inclusive democracy

Mr. Amit Dhoju
Sr. Software Engineer

Day 3: Database Fundamentals

- Database Fundamentals
- MySQL Implementation
- POST,PUT & PATCH
- Security & Design

Section 1: Database Fundamentals

What is a Database?

- An organized collection of data that can be easily accessed, managed, and updated.
- Types
  - SQL Databases(Structured Query Language)
    - MySQL, PostgreSQL, Oracle
  - NoSQL Databases(No SQL)
    - MongoDB, CouchDB
  - In-Memory Databases
    - Redis, Memcached

AMIT DHOJU

What is a Database?

- An organized collection of data that can be easily accessed, managed, and updated.
- Types
  - SQL Databases(Structured Query Language)
    - MySQL, PostgreSQL, Oracle
  - NoSQL Databases(No SQL)
    - MongoDB, CouchDB
  - In-Memory Databases
    - Redis, Memcached

AMIT DHOJU

SQL Databases (Relational)

- Store data in tables (rows & columns).
- Use SQL for queries.

- Examples: MySQL, PostgreSQL, Oracle.

- Pros: Strong consistency, powerful querying, ACID transactions.
- Cons: Less flexible for unstructured data.

AMIT DHOJU

NoSQL Databases (Non-relational)

- Store data in various formats: key-value, document, graph, wide-column.
- Examples: MongoDB, CouchDB.
- Pros: Flexible schema, horizontal scaling.
- Cons: Weaker consistency (eventual consistency in some cases).

AMIT DHOJU

In-Memory Databases

- Store data in RAM for ultra-fast access.
- Examples: Redis, Memcached.
- Pros: Extremely fast, good for caching.
- Cons: Data loss if not persisted to disk.

Which one to choose?

- **Use SQL** when data is highly structured & requires complex queries.
- **Use NoSQL** when data is semi-structured, unstructured, or requires scalability.
- **Use In-Memory** for performance-critical caching and real-time data.

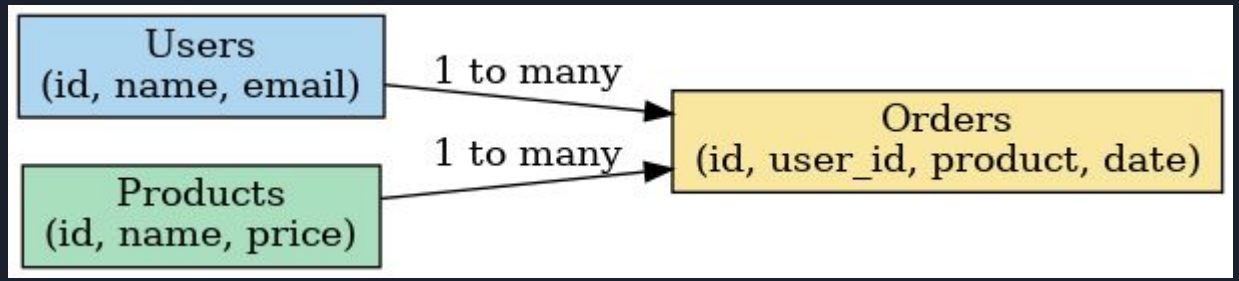
AMIT DHOJU

Section 2: Database Design

Database Design

- Use ER Diagrams to visualize entities and relationships.

- Example:
  - Users table → has many → Orders table.
  - Products table → is in many → Orders table
  - Orders table → belongs to → Users table.



AMIT DHOJU

Section 3: MySQL Implementation

Database: Create Schema

- **Understand Requirements**
  - Identify what data you need to store (e.g., students, colleges).
  - Clarify relationships between data (one-to-many, many-to-many).
- **Identify Entities & Attributes**
  - Entities = tables (e.g., Students, Colleges)
  - Attributes = columns (e.g., name, email)

Database: Create Schema

- **Define Primary Keys (PK)**
  - Unique identifiers for each record (e.g., student_id, college_id).

- **Define Relationships & Foreign Keys (FK)**
  - Link tables (e.g., Orders.user_id → Users.user_id).

AMIT DHOJU

Database: Create Schema

- **Define Primary Keys (PK)**
  - Unique identifiers for each record (e.g., user_id, order_id).

- **Define Relationships & Foreign Keys (FK)**
  - Link tables (e.g., Orders.user_id → Users.user_id).

AMIT DHOJU

Database: Create Schema

```
CREATE SCHEMA `training`
DEFAULT CHARACTER SET
utf8mb4 ;
```

AMIT DHOJU

Database: Create tables

```sql
CREATE TABLE colleges (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    address VARCHAR(255)
);

CREATE TABLE students (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    unique_id VARCHAR(50) UNIQUE NOT NULL,
    college_id INT,
    FOREIGN KEY (college_id) REFERENCES colleges(id)
);
```

Database Setup with MySQL

- Install MySQL locally or use cloud DB services.
- Use MySQL Workbench for GUI-based management.
- Connecting MySQL to Node.js/Express
  - `***npm install mysql***`
- CRUD operations

AMIT DHOJU

Connecting MySQL to Node.js

```
const mysql = require('mysql');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'password',
  database: 'test_db'
});

connection.connect(err => {
  if (err) throw err;
  console.log("Connected to MySQL!");
});
```

AMIT DHOJU

CRUD Operations : Create

- Create new record in database

- connection.query('INSERT INTO users (name, email) VALUES (?, ?)', ['John', 'john@example.com']);

CRUD Operations : Read

- Read data from database

- connection.query('SELECT * FROM users', (err, results) => {
- console.log(results);
- });

AMIT DHOJU

CRUD Operations : Update

- Update existing record in database.

- connection.query('UPDATE users SET name=? WHERE id=?', ['Mike', 1]);

CRUD Operations : Delete

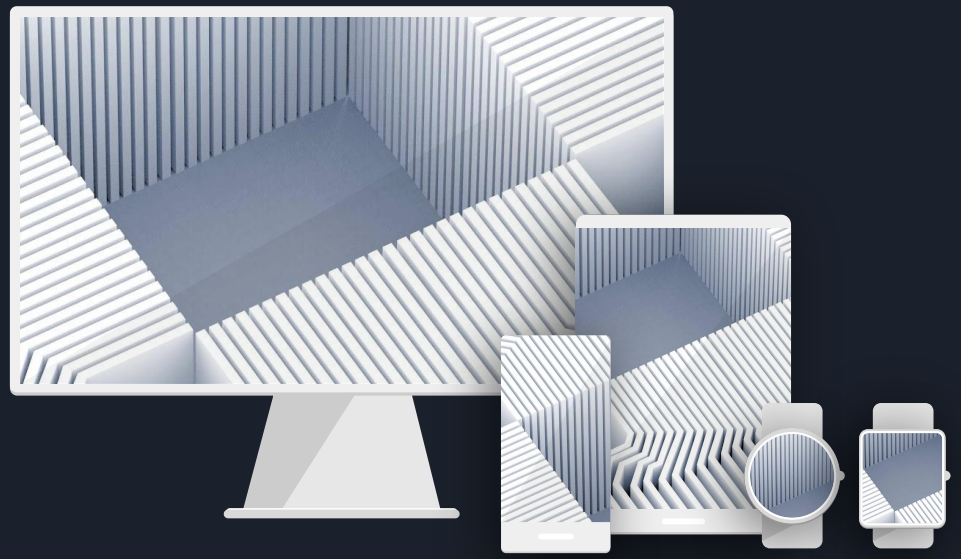
- Delete existing record by id

- connection.query('DELETE FROM users WHERE id=?', [1]);

AMIT DHOJU

Hands-On Exercise:

Express app:
Apply on express app



AMIT DHOJU

Any Questions??

La yo chai
sodhnu parla

AMIT DHOJU

Section 2:
Security & Design

Security Threats

- SQL Injection Prevention
  - Always use prepared statements or parameterized queries.

- XSS Prevention
  - Sanitize input before rendering on the client.
  - Use libraries like helmet in Express.

Security Threats

- SQL Injection Prevention
  - Always use prepared statements or parameterized queries.

- XSS Prevention
  - Sanitize input before rendering on the client.
  - Use libraries like helmet in Express.

SQL Injection Prevention

```
// DO NOT USE - Vulnerable to SQL Injection
app.get('/user', (req, res) => {
  const userId = req.query.id; // e.g. 1 OR 1=1
  const query = `SELECT * FROM users WHERE id = ${userId}`;

  connection.query(query, (err, results) => {
    if (err) throw err;
    res.json(results);
  });
});
```

AMIT DHOJU

SQL Injection Prevention

📌 If a user sends:

bash

```
/user?id=1 OR 1=1
```

The query becomes:

sql

```
SELECT * FROM users WHERE id = 1 OR 1=1;
```

This returns **all users**, bypassing authentication.

AMIT DHOJU

SQL Injection: Safe Implementation

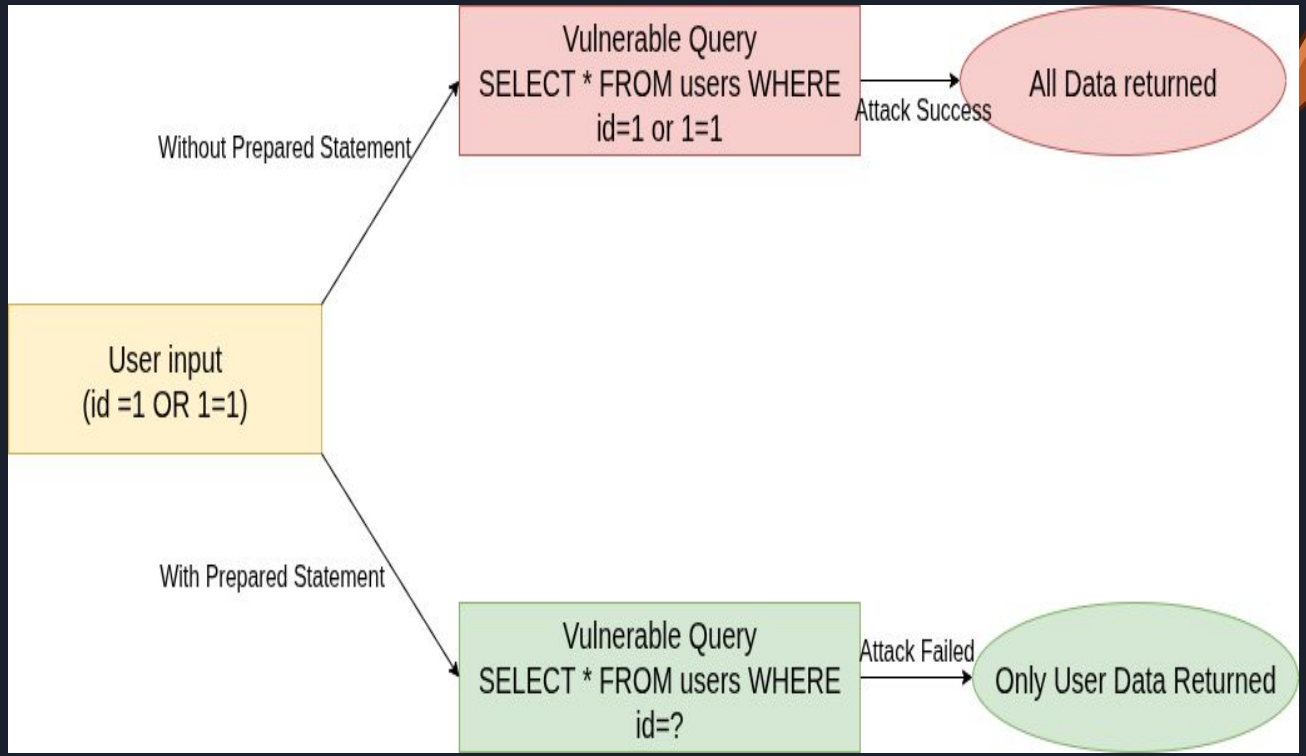
```
// SAFE - Using parameterized query
app.get('/user', (req, res) => {
  const userId = req.query.id;
  const query = 'SELECT * FROM users WHERE id = ?';

  connection.query(query, [userId], (err, results) => {
    if (err) throw err;
    res.json(results);
  });
});
```

Why safe?

- The `?` is replaced with escaped values — SQL code injection is impossible.

AMIT DHOJU

SQL Injection: Safe Implementation



AMIT DHOJU

XSS (Cross-Site Scripting)

🔴 **Vulnerable Example:**

```js
app.get('/search', (req, res) => {
  const term = req.query.q;
  res.send(`<h1>Results for: ${term}</h1>`);
});
```

📌 If someone sends:

```javascript
/search?q=<script>alert('Hacked')</script>
```

The browser executes the script.

AMIT DHOJU

XSS: Safe Implementation

```
const escapeHtml = (unsafe) =>
  unsafe.replace(/&/g, "&amp;")
        .replace(/</g, "&lt;")
        .replace(/>/g, "&gt;")
        .replace(/"/g, "&quot;")
        .replace(/'/g, "&#039;");

app.get('/search', (req, res) => {
  const term = escapeHtml(req.query.q);
  res.send(`<h1>Results for: ${term}</h1>`);
});
```

AMIT DHOJU

XSS: Safe Implementation

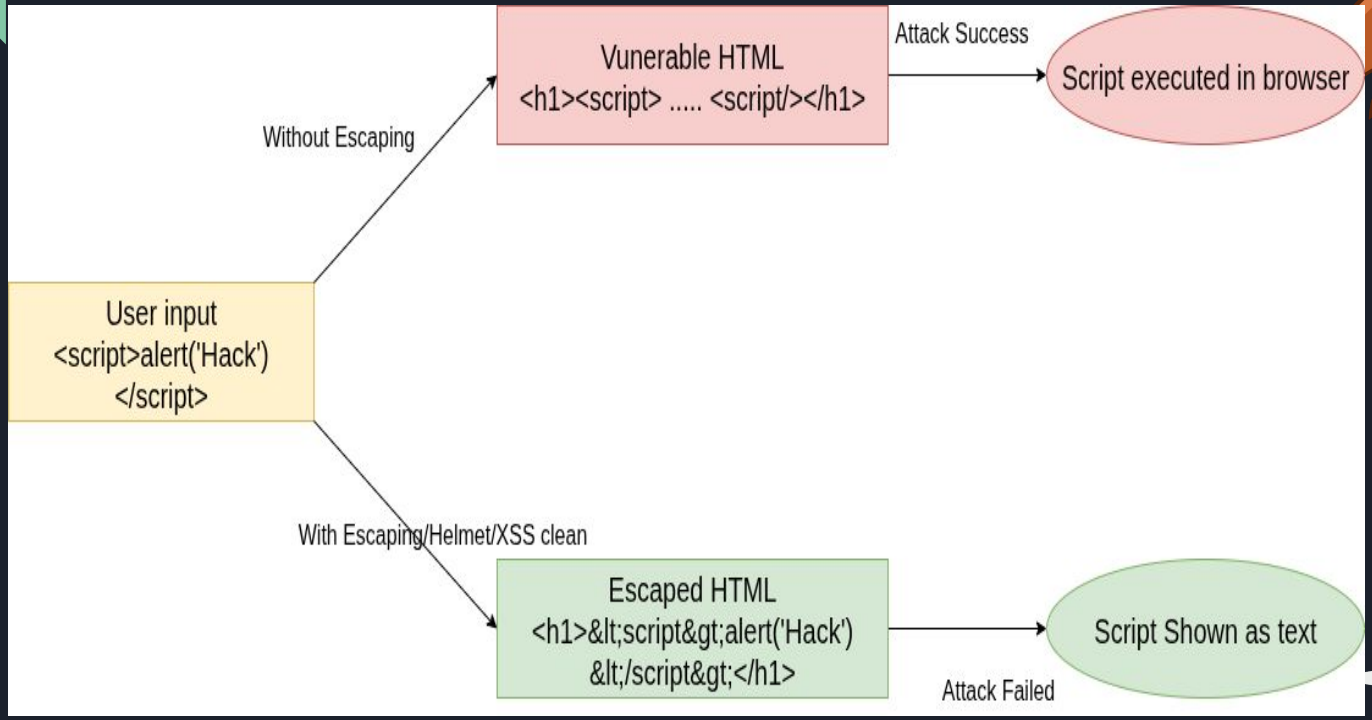
-

```bash
npm install helmet xss-clean
```

```js
const helmet = require('helmet');
const xss = require('xss-clean');

app.use(helmet()); // Security headers
app.use(xss());    // Prevent XSS
```

AMIT DHOJU

XSS: Safe Implementation



AMIT DHOJU

Any Questions??

La yo chai
sodhnu parla

AMIT DHOJU

That’s all for today🫡

Thank you for your time