1. O que é o Express.js e quais suas principais vantagens?

O **Express.js** é um **framework web para Node.js** que facilita a criação de servidores e APIs.

Vantagens:

- Código simples e rápido
- Roteamento fácil (GET, POST, etc.)
- Uso de middlewares
- Grande comunidade e suporte

2. Quais são os quatro principais tipos de rotas HTTP?

1. **GET** – Busca ou lê dados do servidor.
   Ex: listar usuários.
2. **POST** – Envia dados para o servidor.
   Ex: criar um novo usuário.
3. **PUT** – Atualiza dados existentes.
   Ex: editar um usuário.
4. **DELETE** – Remove dados do servidor.
   Ex: deletar um usuário.

3. Explique o conceito de middleware no Express.js

Um **middleware** no Express.js é uma **função que executa entre a requisição (req) e a resposta (res)**.

Ele pode:

- Modificar a requisição ou resposta
- Finalizar a resposta
- Chamar o próximo middleware com next()

4. Como criar uma conexão básica entre Node.js e MySQL?

```
const mysql = require('mysql2');

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

connection.connect(err => {
  if (err) {
    console.error('Erro na conexão:', err);
    return;
  }
  console.log('Conectado ao MySQL!');
});
```

5. O que é um prepared statement e como ele protege contra SQL Injection?

Um **prepared statement** é uma consulta SQL com **parâmetros separados dos dados**. Ele prepara o comando antes de inserir os valores.

Proteção contra SQL Injection:

Como os dados são enviados **separadamente da instrução SQL**, o banco de dados **não interpreta entradas maliciosas como código**, apenas como valores.

6. Qual é a estrutura básica de um servidor Express.js?

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

```
const app = express();


// Rota simples
app.get('/', (req, res) => {
  res.send('Olá, mundo!');
});


// Inicia o servidor
app.listen(3000, () => {
  console.log('Servidor rodando na porta 3000');
});
```

7. Explique a diferença entre rotas globais e rotas específicas

Rotas globais: São aplicadas a **todas as requisições**, independentemente do caminho ou método. Usam geralmente app.use().

Rotas específicas: Respondem a um **caminho e método definidos**, como GET /users ou POST /login.

8. Três boas práticas ao manipular dados em bancos de dados:

1. **Usar prepared statements**
   Protege contra **SQL Injection** ao separar dados do comando SQL.
2. **Validar dados antes de salvar**
   Garante que só informações corretas e esperadas sejam enviadas ao banco.
3. **Tratar erros sempre**
   Use try/catch ou callbacks com verificação de erro para evitar falhas silenciosas ou travamentos.

9. Por que é importante usar variáveis de ambiente em aplicações Node.js?

Variáveis de ambiente permitem **guardar configurações e dados sensíveis fora do código-fonte**, como senhas, portas e URLs.

Vantagens:

- **Segurança**: evita expor credenciais no código.
- **Flexibilidade**: muda configurações sem alterar o código.
- **Portabilidade**: facilita usar a mesma aplicação em ambientes diferentes (dev, teste, produção).

10. Como tratar erros em consultas SQL dentro do Node.js?

Você pode tratar erros usando **callbacks com verificação de erro** ou **blocos try/catch** (quando usar async/await com mysql2/promise).

Exemplo com callback:

```
connection.query('SELECT * FROM usuarios', (err, results) => {
  if (err) {
    console.error('Erro na consulta:', err.message);
    return;
  }
  console.log(results);
});
```

Exemplo com async/await:

```
try {
  const [rows] = await connection.execute('SELECT * FROM usuarios');
  console.log(rows);
} catch (err) {
  console.error('Erro na consulta:', err.message);
}
```