Qual seria a estrutura básica do código para implementar essa API? Apresente um exemplo de código.

2. Banco de Dados   
  
a) Que banco de dados você recomendaria para essa aplicação? Justifique sua escolha.   
  
b) Escreva um modelo de esquema usando Sequelize ou Mongoose para representar as entidades Cliente, Produto e Compra.

3. Autenticação e Segurança   
  
a) Como você implementaria autenticação para os clientes? Utilize JWT em seu exemplo.   
  
b) Como garantiria a segurança das senhas no banco de dados? Apresente um trecho de código ilustrando sua resposta.

4. Boas Práticas e Testes   
  
a) Quais são as boas práticas para organizar o código de uma API Node.js?   
  
b) Como você implementaria testes para os endpoints da API? Forneça um exemplo usando Jest ou Supertest.

5. Desempenho e Escalabilidade   
  
a) Quais estratégias você utilizaria para melhorar o desempenho da API?   
  
b) Como garantiria que a API possa escalar para atender um grande número de requisições?

Respostas:  
  
  
**2. Banco de Dados**

**a) Recomendação:**  
**PostgreSQL** com **Sequelize**. É robusto, relacional, possui suporte a transações e integra bem com Node.js via ORM.  
  
**b) Modelos com Sequelize:**

// Cliente.js

module.exports = (sequelize, DataTypes) => {

return sequelize.define('Cliente', {

nome: DataTypes.STRING,

email: DataTypes.STRING,

senha: DataTypes.STRING

});

};

// Produto.js

module.exports = (sequelize, DataTypes) => {

return sequelize.define('Produto', {

nome: DataTypes.STRING,

descricao: DataTypes.STRING,

preco: DataTypes.FLOAT

});

};

// Compra.js

module.exports = (sequelize, DataTypes) => {

return sequelize.define('Compra', {

cliente\_id: DataTypes.INTEGER,

produto\_id: DataTypes.INTEGER

});

};

**3. Autenticação e Segurança**

**a) Autenticação com JWT:**const jwt = require('jsonwebtoken');

function gerarToken(cliente) {

return jwt.sign({ id: cliente.id, email: cliente.email }, 'secreto', { expiresIn: '1h' });

}

**b) Segurança das senhas com bcrypt:**  
  
const bcrypt = require('bcrypt');

cliente.senha = await bcrypt.hash(cliente.senha, 10);

const senhaValida = await bcrypt.compare(senhaDigitada, cliente.senha);  
  
  
**4. Boas Práticas e Testes**

**a) Boas práticas:**

* Separar responsabilidades (MVC)
* Usar variáveis de ambiente com dotenv
* Validar entradas com Joi ou express-validator
* Documentar com Swagger
* Tratar erros com middlewares

**b) Testes com Jest:**// clienteController.test.js

const { criarCliente } = require('../controllers/clienteController');

const { Cliente } = require('../models');

jest.mock('../models');

describe('criarCliente', () => {

it('deve criar um cliente com sucesso', async () => {

const req = {

body: { nome: 'Maria', email: 'maria@email.com', senha: '123456' }

};

const res = { status: jest.fn().mockReturnThis(), json: jest.fn() };

Cliente.create.mockResolvedValue({ id: 1, ...req.body });

await criarCliente(req, res);

expect(res.status).toHaveBeenCalledWith(201);

expect(res.json).toHaveBeenCalledWith(expect.objectContaining({ nome: 'Maria' }));

});

});

**5. Desempenho e Escalabilidade**

**a) Estratégias de desempenho:**

* Cache com Redis
* Paginação em endpoints
* Indexação no banco

**b) Escalabilidade:**

* Balanceamento de carga (ex: Nginx)
* Microsserviços para módulos independentes
* Banco de dados replicado