Nome: Viviane J. Santos **Docente:** Ivan F. Borges

Exercício 2: Linguagem de Programação para API

Parte Teórica:

1. Linguagem de Programação para APIs: Qual a importância de escolher a linguagem de programação correta para o desenvolvimento de APIs e quais fatores devem ser considerados ao fazer essa escolha?
R: A escolha da linguagem correta dependerá do tipo de projeto que o programador deseja desenvolver. Ela depende do ecossistema da aplicação, requisitos de desempenho, e familiaridade da equipe com a linguagem.

2. Funcionalidades para APIs: Quais são algumas das funcionalidades essenciais que uma linguagem de programação deve oferecer para facilitar o desenvolvimento de APIs?
R:

- **Manipulação de Requisições HTTP** => Bibliotecas e Frameworks e Middlewares.
- **Suporte a JSON e XML** => JSON (JavaScript Object Notation) e XML (Extensible Markup Language).
- **Conectividade com Bancos de Dados** => Drivers e ORMs e Tipos de Bancos de Dados.
- **Gerenciamento de Sessões e Autenticação** => Cookies e Sessions, JWT (JSON Web Tokens) e OAuth.
- **Suporte a WebSockets** => Comunicação em Tempo Real e Linguagens com Suporte a WebSockets.

3. Técnicas de Depuração: Cite duas técnicas de depuração que podem ser usadas para identificar e corrigir erros em APIs. Explique brevemente como cada uma delas funciona.
R: Logs de Servidor => Ferramentas de Logging: Ferramentas como Winston para Node.js oferecem grande flexibilidade, permitindo que logs sejam gravados em diferentes destinos (arquivos, banco de dados, console) e formatos (JSON, texto simples).
Debugging em Tempo Real => Ferramentas de Debugging: Depuradores de código permitem pausar a execução do programa e inspecionar o estado do aplicativo em pontos específicos. No Node.js, o Node.js Inspector é integrado e permite depuração diretamente no Chrome DevTools.

4. Documentação de APIs: Por que é importante documentar APIs e quais são as boas práticas ao criar a documentação?
R: A documentação de APIs também serve como um local para os desenvolvedores resolverem dúvidas sobre sintaxe ou funcionalidade. Além disso, uma documentação excelente tende a aumentar a adoção da API, já que a experiência de uso torna-se muito mais fácil e amigável aos desenvolvedores.
As melhores práticas de elaboração desse documento são:

- Apresentar bons exemplos;
- Fazer uma boa categorização;
- Centralizar as informações;
- Investir em interatividade;
- Testar a documentação;
- Elaborar bem as mensagens de erro.

5. Técnicas de Programação e Controle: Explique o que é modularização e por que é importante no desenvolvimento de APIs. Dê um exemplo de como a modularização pode ser aplicada em um projeto.
R: A modularização permite que uma parte do código possa ser alterada ou atualizada sem que todo o código já desenvolvido seja alterado. Dessa maneira, fica muito mais fácil atualizar um software ou corrigir um problema específico em um sistema.
Ex.: Separando a lógica de autenticação da lógica de manipulação de dados, isso permite modificar uma sem afetar a outra.

6. Frameworks: Nomeie dois frameworks comuns para desenvolvimento de APIs em Node.js e Python, respectivamente, e explique por que eles são populares.
R: Node.js =

- **Express.js:** É um dos frameworks mais populares para o desenvolvimento de aplicações web em Node.js. Ele é conhecido por sua simplicidade e flexibilidade, permitindo a criação rápida e fácil de APIs e servidores web. Além de ser altamente recomendado para projetos que exigem uma abordagem minimalista e modular, onde a liberdade de escolha é valorizada.

- **Nest.js:** É um framework que combina o poder do TypeScript com a arquitetura modular e orientada a serviços. Ele promove a organização e a escalabilidade do código, facilitando a construção de aplicações backend robustas e bem estruturadas. Além de ser uma excelente escolha para projetos que exigem uma arquitetura escalável e uma base de código bem organizada.

Python =

- **FastAPI** e **Flask** são dois frameworks para desenvolvimento de APIs web em Python. Ambos são amplamente utilizados e possuem características e vantagens próprias.
- **Flask** é conhecido por sua simplicidade e flexibilidade, enquanto **FastAPI** oferece um desempenho excepcional e uma ótima integração com a concepção OpenAPI. A escolha entre os dois depende dos requisitos específicos do projeto e das preferências pessoais do desenvolvedor.

7. Status de Respostas: Qual é a diferença entre os códigos de status HTTP 200 e 404? Dê um exemplo de situação em que cada um deles seria usado.
R: Códigos de status HTTP indicam o resultado de uma operação. Eles informam ao cliente se a requisição foi bem-sucedida ou se ocorreu algum erro.
HTTP 200 = `200 OK`: solicitação bem-sucedida.

A requisição foi processada com sucesso. Usado principalmente para requisições GET, POST (quando não há criação de recurso), PUT, e DELETE quando a operação foi bem-sucedida.
Ex.: res.status(200).json({ message: " Solicitação bem-sucedida"});

HTTP 400 = 400 Solicitação Incorreta: argumento inválido (Carga da solicitação inválida).
A requisição foi malformada ou contém parâmetros inválidos. O servidor não pode processar a requisição devido a um erro do cliente. (sintaxe de requisição mal formada, enquadramento de mensagem de requisição inválida ou requisição de roteamento enganosa)
Ex.: res.status(400).json({ error: "Data de entrada inválida"});

8. Routes: O que são rotas (routes) em uma API? Dê um exemplo de rota simples usando o framework Express.js.
R: É um método para mapear URLs para funções específicas que processam requisições e retornam respostas dessas requisições.
Ex.:
```
app.get('/book/:id', (req, res) => {
        const book = books.find(b => b.id === parseInt(req.params.id));
        if (!book) return res.status(400).send("Book not found);
        res.json(book);
});
```

9. Tratamento de Exceções: Como o uso de blocos try/catch e middlewares de erro pode melhorar a robustez de uma API? Dê um exemplo em que isso seria aplicado.
R: Usado sempre onde se espera que uma exceção possa ocorrer em um trecho específico de código e deseja lidar com essa exceção de forma apropriada, sem que o programa pare de funcionar abruptamente. Isso melhora a robustez do código e evita que o programa quebre de forma inesperada.
Ex.:
```
app.get('/books/:id', async (req, res) {
        try {
        const book = await Book.findById(req.params.id);
        if (!book) return res.status(404).send('Book not found');
        res.json(book);
        } catch (err) {
        res.status(500).send('An error occurred');
}
});
```

10. Técnicas de Formato de Comunicação: Qual a diferença entre JSON e XML em termos de uso em APIs? Cite uma vantagem de usar JSON sobre XML.
R: O JSON usa uma estrutura semelhante a um mapa com pares de valores-chave. O XML armazena dados em uma estrutura de árvore com namespaces para diferentes categorias de dados.
A sintaxe do JSON é mais compacta e fácil de ler e escrever.

A sintaxe do XML substitui alguns caracteres por referências de entidades, tornando-a mais detalhada.
A vantagem do JSON sobre o XML são: Simplicidade, Performance, Flexibilidade.
APIs web modernas, especialmente RESTful, utilizam JSON para a maioria das operações de comunicação.

11. Autenticação: Explique a diferença entre Basic Auth e OAuth em termos de segurança e uso.
R: Basic Auth = Envia credenciais de nome de usuário e senha codificadas em base64 no cabeçalho HTTP da requisição.
É simples, mas não seguro sem HTTPS, pois as credenciais
podem ser facilmente decodificadas.

OAuth = É um padrão aberto de autorização que permite que aplicativos acessem recursos em nome de um usuário, sem expor as credenciais do usuário. Ele usa tokens de acesso e é amplamente usado em APIs de redes
sociais, serviços de email e outras aplicações que requerem acesso a dados de terceiros.

12. Cookies, Sessions, Token, JWT: Qual é a principal vantagem de usar JWT (JSON Web Tokens) em comparação com sessões tradicionais baseadas em cookies?
R: São tokens compactos que podem ser usados para autenticação e autorização.
Eles são compostos de três partes: header, payload, e signature.
JWTs são autossuficientes, significando que contêm todas as informações necessárias para verificar a identidade do usuário, eliminando a necessidade de armazenar o estado no servidor.

Parte Prática:

Objetivo: Implementar uma API simples usando Node.js e Express que demonstre o uso de rotas.

Instruções: Crie uma API com as seguintes funcionalidades:

- **Rota GET /books que retorna uma lista de livros.**
- **Rota GET /books/:id que retorna um livro específico pelo ID.**
- **Rota POST /books que adiciona um novo livro.**
- **Rota DELETE /books/:id que remove um livro pelo ID.**

Desafio: Implementar na API: tratamento de exceções, autenticação com JWT e manipulação de JSON.