Aula Node

## **O que é NPM ?**

O NPM (Node Package Manager, em tradução literal, Administrador de Pacotes para o Node) é um registro de códigos para o desenvolvimento em JavaScript, seu uso é gratuito. Desenvolvedores de **código livre**/**código aberto** publicam seus programas no NPM para compartilhar-los, isso proporcionou com que o NPM se torna-se o maior registro de códigos do mundo. Muitas dessas ferramentas tornaram-se extremamente populares ao ponto de ser um requisito profissional para muitas empresas, como: Express, React, React Native e Angular.

## **Como funciona**

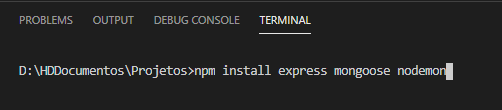
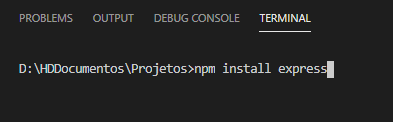
NPM consegue controlar os pacotes/dependências locais ou globais de um determinado projeto. Quando usado em um projeto local, o NPM consegue instalar todas as dependências desse projeto com apenas um comando, “*npm install*”, como veremos com mais calma futuramente, através do arquivo JSON “*package.json*”, responsável por registrar todos os dados/meta-dados de nosso projeto, como: nome do projeto, autor, descrição, versão e dependências. Dentro desse arquivo JSON, cada dependência possui um intervalo de versões válidas, definida por um *semantic versioning scheme* (esquema de versão semântico), isso permite com que os desenvolvedores atualizem suas dependências evitando que elas quebrem seus projetos.

# **Como usar**

Há diversas maneiras de se começar, primeiramente é interessante criar o arquivo *package.json,* para isso é necessário você abrir o terminal, ir até a pasta do projeto e digitar ***npm init***

1. Abra o terminal, navegue até a pasta em que você criará seu projeto, usando o comando “*cd <pasta>*” e digite *npm init*
2. Alternativamente, você também pode abrir o visual studio, ir em file > abrir pasta, selecionar a pasta, e em seguida abrir o terminal (terminal > novo terminal) do visual studio e digitar ***npm init.***

Feito isso, basta você escolher as dependências que deseja e digitar ***npm install <dependencia>***, você também pode instalar múltiplas dependências ao mesmo tempo, para isso, separe os nomes das dependência com um espaço, veja alguns exemplos:



Na segunda figura, vemos um exemplo de instalações múltiplas: express, mongoose e nodemon.

# **Versionamento Semântico (Semantic Versioning)**

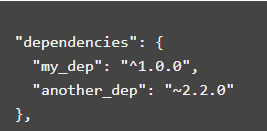
Para manter o ecossistema do JavaScript saudável, confiável e seguro, os desenvolvedores de pacotes/dependências são orientados à seguir algumas regras de versão toda vez que atualizam/alteram alguma coisa em seus pacotes. Isso é mostrado no *package.json*, exemplos:

* 1.0.x : Correão de bugs – Patch -, ex: 1.0.0 para 1.0.1
* 1.x.x : Novas *features* ***compatíveis*** – Minor -, ex: 1.0.1 para 1.1.0
* X.x.x : Mudanças ***incompatíveis*** com códigos anteriores – Major -, Ex: 1.1.0 para 2.0.0

Você pode diretamente especificar quais tipos de atualizações você irá aceitar em seus pacotes, seguindo as seguinte sintaxe:

* Patch releases: **1.0** ou **1.0.x** ou **~1.0.4**
* Minor releases: **1** ou **1.x** ou **^1.0.4**
* Major releases: **\*** ou **x**

Exemplo:



# Express, simples e poderoso

Algumas tarefas comuns no desenvolvimento web não são suportadas diretamente pelo Node. Se você quiser que a sua aplicação possua diferentes verbos HTTP (por exemplo GET, POST, DELETE... veremos isso com mais detalhe em futuras aulas), que gerencie requisições de diferentes URLs ("**rotas**"), apresente arquivos estáticos ou utilize ***templates*** para mostrar as respostas (response) de maneira dinâmica, você não terá muita praticidade usando apenas o Node. Você terá duas opções. Escrever o código por conta própria ou então evitar todo esse trabalho de reinventar a roda ao utilizar um framework.

O ***Express.js***, ou simplesmente ***Express***, é um framework JavaScript de código aberto rápido, flexível e minimalista para Node.js. Ele foi criado para desenvolver **aplicações web** e **API** (Application Programming Interface, do Inglês, Interface de Programação de Aplicações). O Express oferece soluções para:

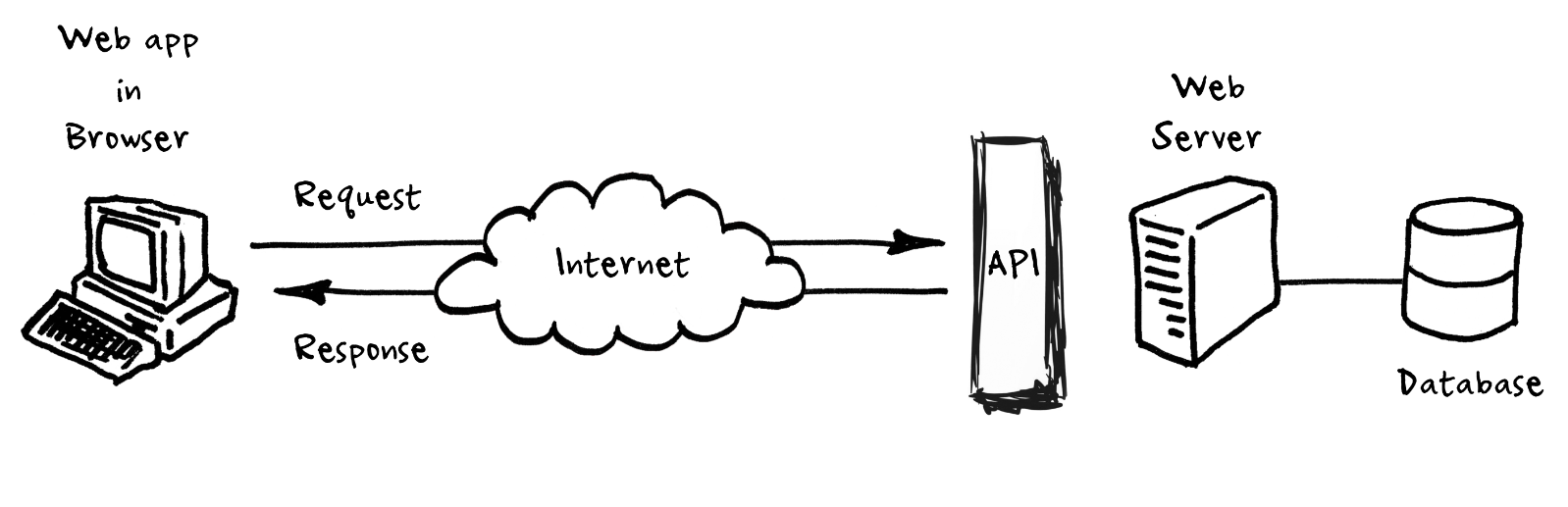
* Gerenciar requisições de diferentes verbos HTTP em diferentes URLs.
* Integrar "***view engines***" para inserir dados nos *templates*.
* Definir as configurações comuns da aplicação web, como a porta a ser usada para conexão e a localização dos modelos que são usados para renderizar a resposta.
* Adicionar novos processos de requisição por meio de "middleware" em qualquer ponto da "fila" de requisições.

# O que é uma API ?

**API** provém do Inglês ***Application Programming Interface***, é um conjunto de [rotinas](https://pt.wikipedia.org/wiki/Sub-rotina) e padrões estabelecidos por um [software](https://pt.wikipedia.org/wiki/Software) para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da [implementação](https://pt.wikipedia.org/wiki/Implementação_de_software) do software, mas apenas usar seus serviços.

De modo geral, a API é composta por uma série de funções acessíveis somente por [programação](https://pt.wikipedia.org/wiki/Linguagem_de_programação), e que permitem utilizar características do software menos evidentes ao utilizador tradicional.

* **Propósito:** O propósito das *APIs* é o de simplificar o desenvolvimento de [programas e aplicações](https://pt.wikipedia.org/wiki/Aplicações), através da [abstração](https://pt.wikipedia.org/wiki/Abstração) dos diversos elementos que compõem um *software*. Isso permite que o desenvolvedor não precise saber detalhadamente a implementação de todas as partes que compõem um sistema, mas sim, precisará saber como utilizar os serviços fornecidos pelas *APIs* de seu interesse e como se dará a interação destas com os outros elementos de seu *software*.

Em resumo, uma **API** é como se fosse uma “*ponte*” que permite que aplicações se comuniquem umas com as outras, ela recebe uma ***requisição*** e retorna uma ***resposta*** de volta para o cliente. Muitas pessoas quando perguntadas o que é uma API, se confundem e dizem que API’s são “*um serviço web que retorna dados como XML ou JSON*”. Isso não é exatamente a API, mas sim seu “produto final”, uma API **não** é **o servidor** e **nem** o **banco de dados**, mas sim o **código** que “*administra”* o ponto de acesso do servidor. 

Leia mais sobre APIs:

* <https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f> (Inglês)
* <https://becode.com.br/o-que-e-api-rest-e-restful/> (Português)

Vídeo para auxiliar:

* <https://www.youtube.com/watch?v=s7wmiS2mSXY> (Inglês)
* <https://www.youtube.com/watch?v=RVIJnOHKOLc> (Português)
* <https://www.youtube.com/watch?v=vGuqKIRWosk> (Português)

Exemplos de APIs:

* <https://postmon.com.br/>
* <https://swapi.dev/>
* <https://www.dnd5eapi.co/>

Fontes:

[https://www.w3schools.com/whatis/whatis\_npm.asp#:~:text=All%20npm%20packages%20are%20defined%20in%20files%20called%20package.&text=The%20content%20of%20package.,definition%20file%3A%20name%20and%20version.](https://www.w3schools.com/whatis/whatis_npm.asp#:~:text=All npm packages are defined in files called package.&text=The content of package.,definition file%3A name and version.)

<https://en.wikipedia.org/wiki/Npm_(software)>

<https://docs.npmjs.com/about-semantic-versioning>

<https://en.wikipedia.org/wiki/Express.js>

https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f

https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/