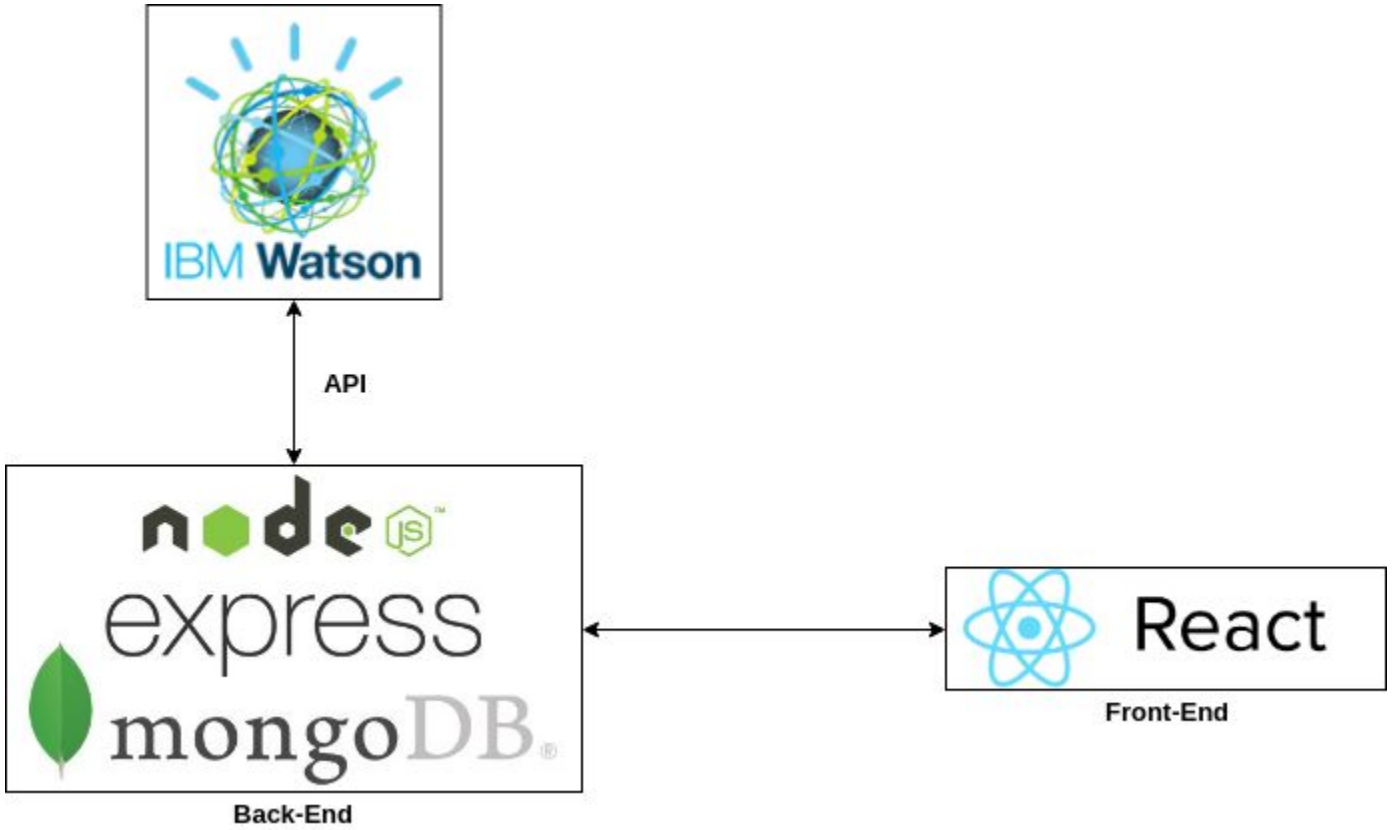
LabRedes de Conhecimento

Chatbots: da teoria ao deploy, com IBM Watson

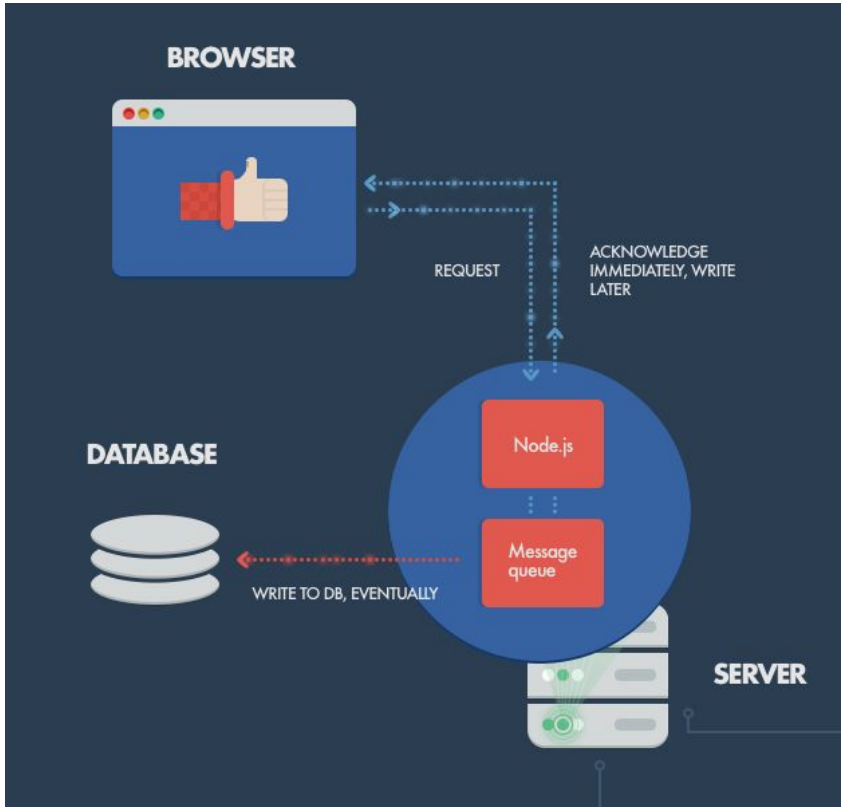
João Paulo de Melo
jpmdik@gmail.com
Tecnólogo em Sistemas para Internet

Aula 05: Criando um serviço para requisição a API Watson

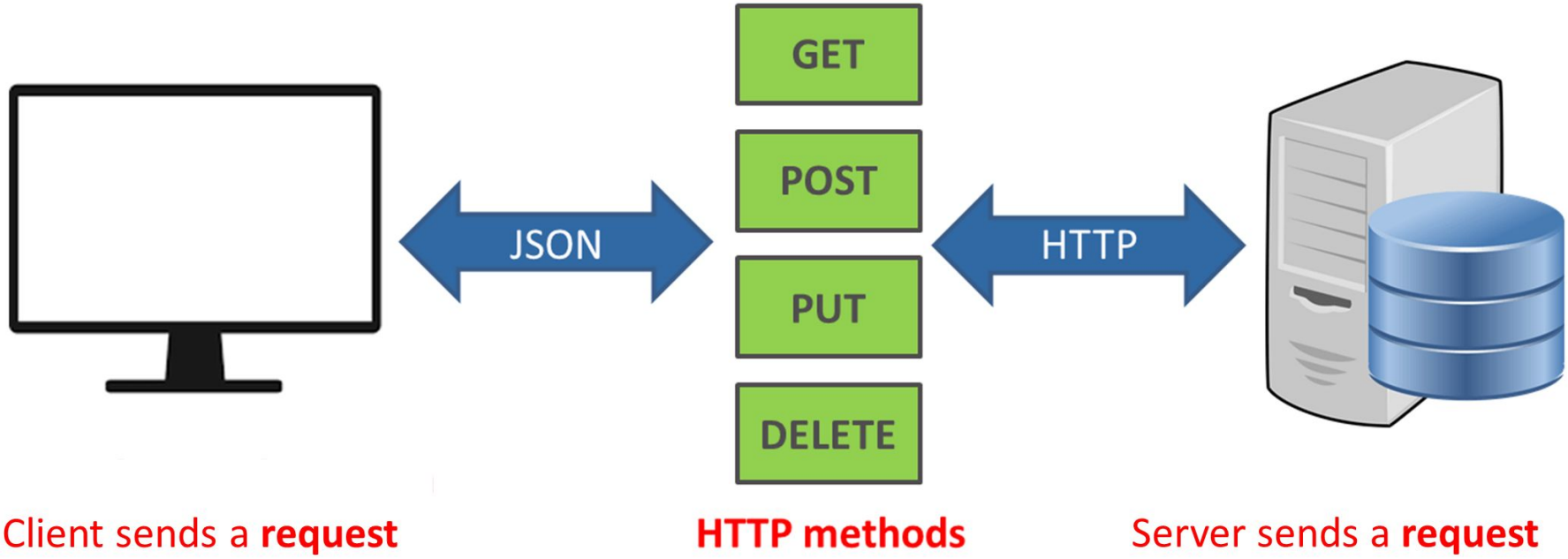
O que vamos construir?



Como funciona a requisição a um back-end



Requisições HTTP



Requisições HTTP

Exemplo: Se Solicitamos a página https://www.google.com/ :

Autoridade: `www.google.com`

Método: `GET`

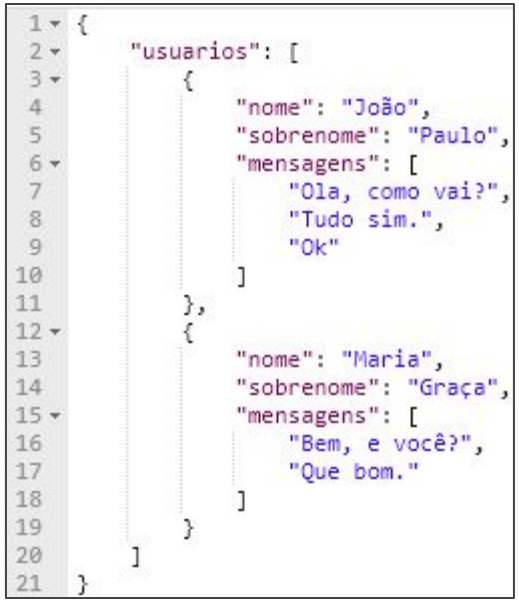
Path: `/`

Schema: `https`

Muitas outras informações podem ser encontradas e utilizadas no cabeçalho do protocolo.

JSON

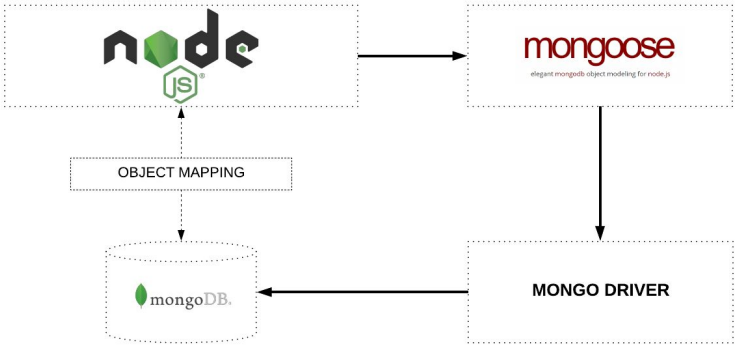
JSON (JavaScript Object Notation) é um formato leve de troca de dados. É fácil para humanos ler e escrever. É fácil para as máquinas analisar e gerar. Ele é baseado em um subconjunto da Linguagem de Programação JavaScript, Padrão ECMA-262 3ª Edição - Dezembro de 1999. JSON é um formato de texto completamente independente do idioma, mas usa convenções que são familiares aos programadores da família C de idiomas, incluindo C , C ++, C #, Java, JavaScript, Perl, Python e muitos outros. Essas propriedades tornam o JSON uma linguagem de intercâmbio de dados ideal.



```
{
    "usuarios": [
        {
            "nome": "João",
            "sobrenome": "Paulo",
            "mensagens": [
                "Ola, como vai?",
                "Tudo sim.",
                "Ok"
            ]
        },
        {
            "nome": "Maria",
            "sobrenome": "Graça",
            "mensagens": [
                "Bem, e você?",
                "Que bom."
            ]
        }
    ]
}
```

MongoDB

MongoDB é um software de banco de dados orientado a documentos livre, de código aberto e multiplataforma, escrito na linguagem C++. Classificado como um programa de banco de dados NoSQL, o MongoDB usa documentos semelhantes a JSON com esquemas.



Iniciando a aplicação

Iniciando a aplicação

- Criar um projeto no Github;
- Clonar o projeto;
- Abrir o projeto utilizando uma IDE de desenvolvimento;
- Inicializar o projeto usando o npm;
- Instalar todas as Dependências;
- Criar o .gitignore;
- Fazer o primeiro versionamento ao Github;

Iniciando a aplicação

Inicializar o projeto usando o npm:

Comando: npm init [-y]

Instalando as dependências:

Comando: npm install --save body-parser@1.15.2 express@4.14.0 express-query-int@1.0.1 mongoose@4.7.0 node-restful@0.2.5 nodemon@1.11.0 pm2@2.1.5 watson-developer-cloud@3.15.1 dotenv@6.2.0

Iniciando a aplicação

Criando o .gitignore:

```
node_modules
*.log
```

loader.js

```
1 const server =    require('./config/server')
2 require('./config/database')
3 require('./config/routes')(server)
```

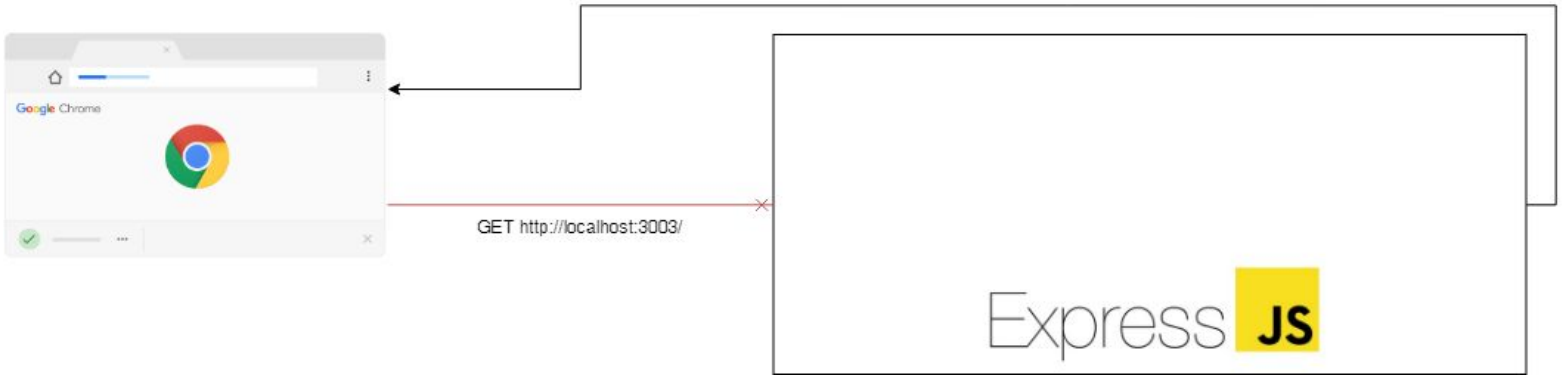
Linha 1: Executa o arquivo que cria o servidor em src/config/server.js juntamente com os *middlewares* que que todas as requisições passarão e retorna a instância do servidor, que é armazenada na constante server

Linha 2: Executa o arquivo src/config/database.js que inicia a conexão com o banco de dados. Não retorna nada porque a conexão é armazenada em uma Promise global.

Linha 3: Executa o arquivo src/config/server.js e passa pra ele a constante que representa o servidor. Para que ele possa criar as rotas. Exemplo: http://localhost:3003/api/produtos GET

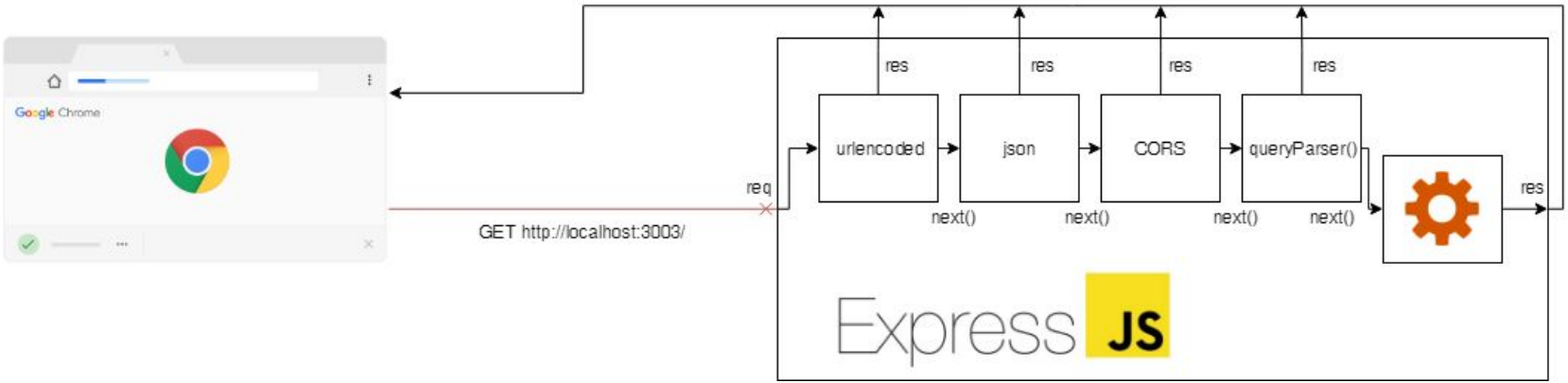
server.js

```
// Criamos a instância inicial do nosso servidor
const server = express()
```



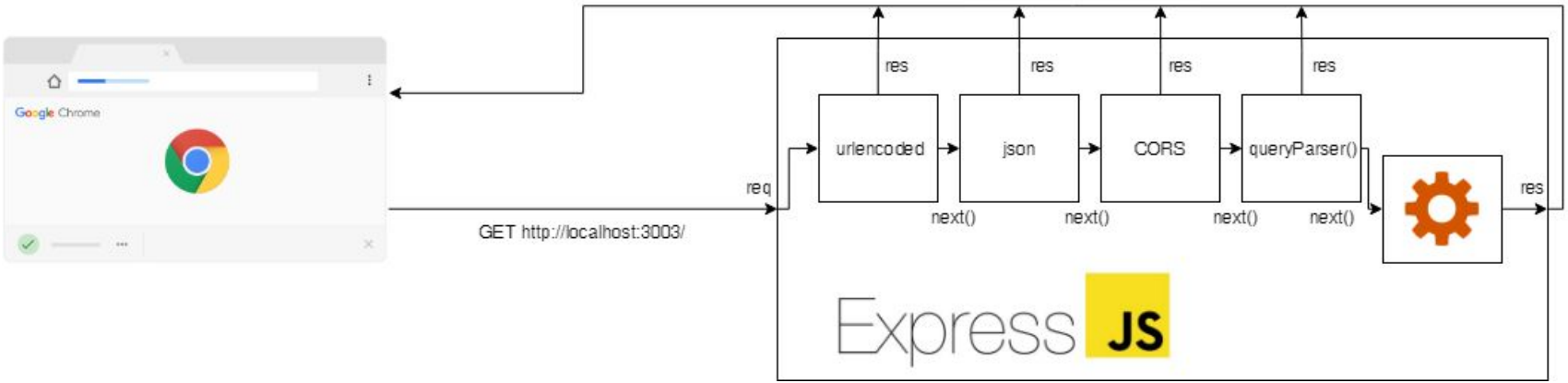
server.js

```
// Aplicamos middlewares ao nosso servidor
server.use(bodyParser.urlencoded({ extended: true })) //Esse middleware trata dados de formulários
server.use(bodyParser.json()) //Esse middleware trata dados de application/json (JSON)
server.use(allowCors) // Esse middleware faz com que as requisições tenham o CORS em seu cabeçalho
server.use(queryParser()) //Converte valores passados string que seriam números para o seu tipo corretamente
```



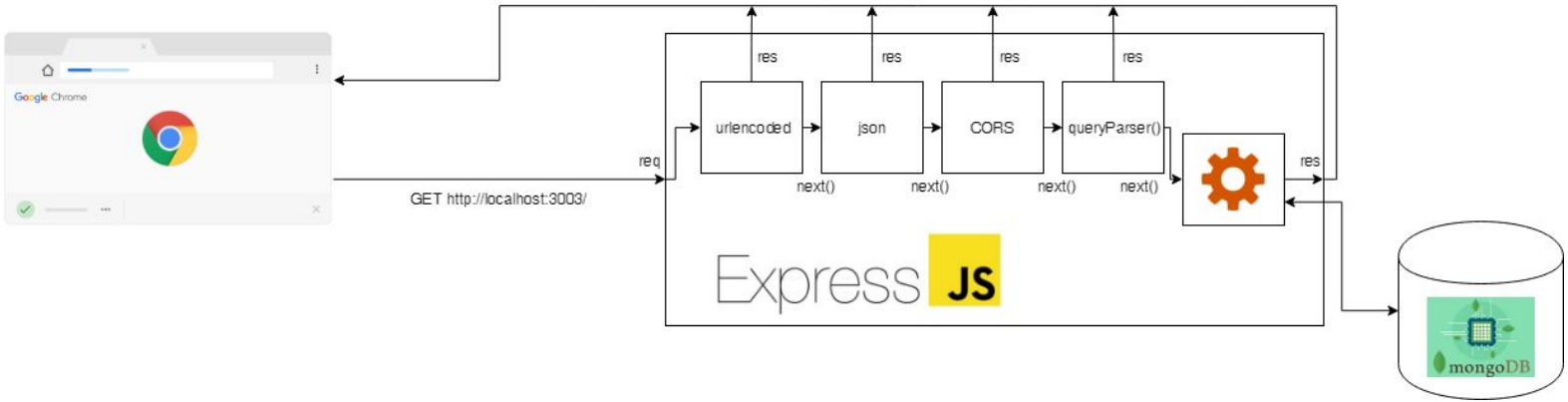
server.js

```
// Rodamos o serviço após aplicar todos os midlewares
server.listen(port, function(){
   console.log(`Backend está rodando na porta ${port}.`);
})
```



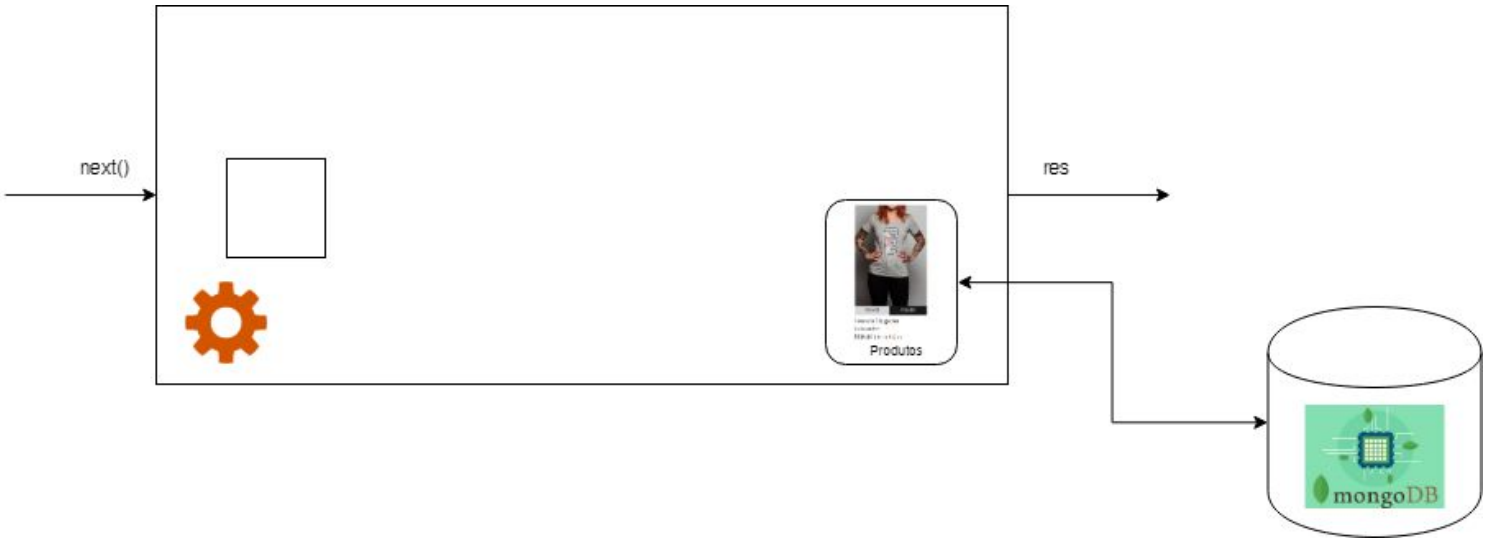
database.js

```
// Cria uma conexão com o banco de dados e exporta.
const mongoose = require('mongoose')
mongoose.Promise = global.Promise
module.exports = mongoose.connect('mongodb://localhost/loja_de_roupas', {useMongoClient: true})
```



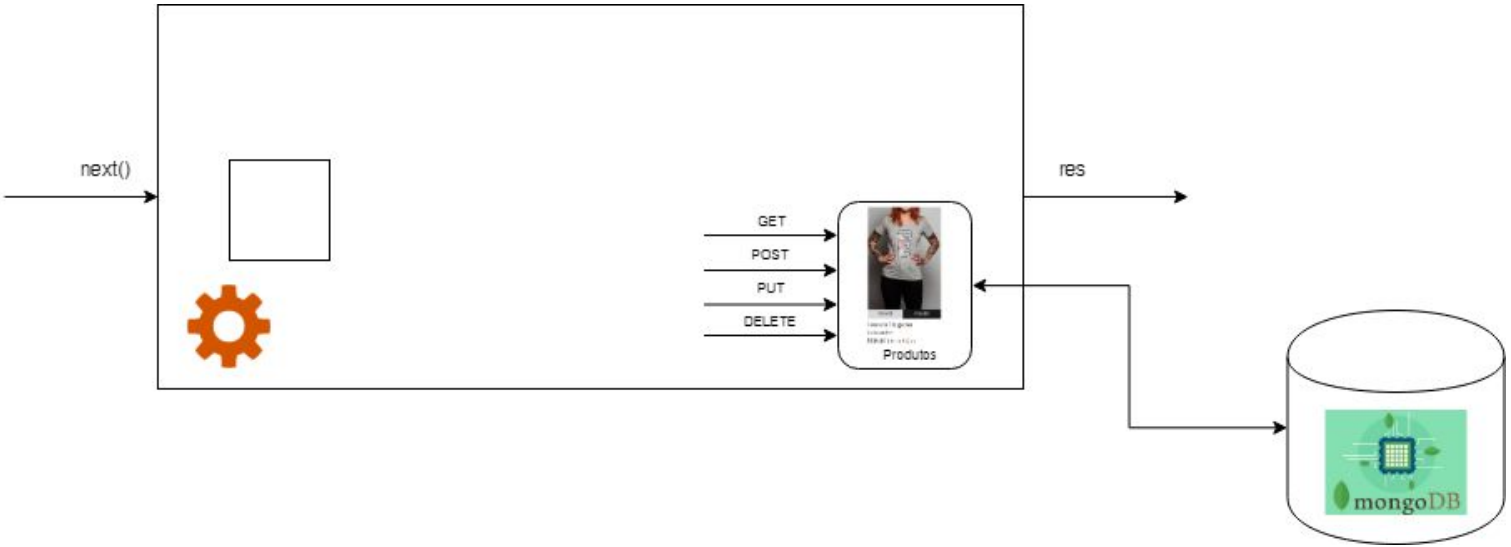
produtos.js

```
const produtosSchema = new mongoose.Schema({
   name: { type: String, required: true },
   imgUrl: { type: String, required: true },
   value: { type: Number, min: 0, required: true },
})
```



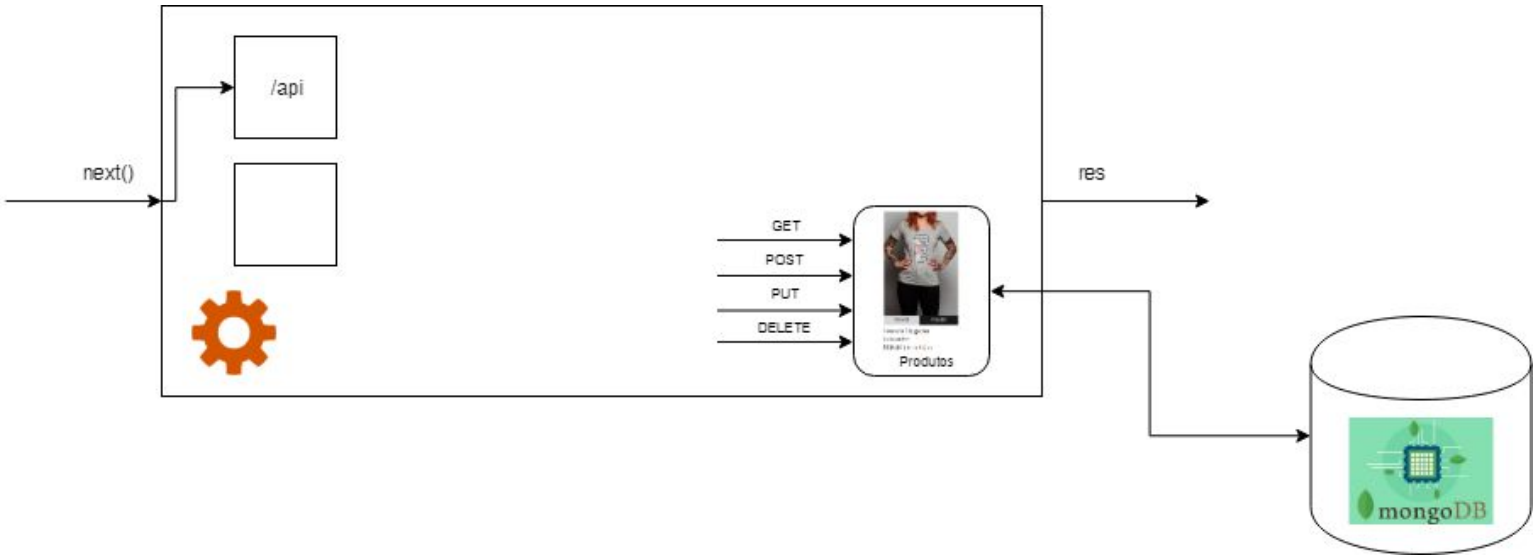
produtosService.js

```
produtos.methods(['get', 'post', 'put', 'delete']) //informa quais tipos de requisições serão permitidos
```



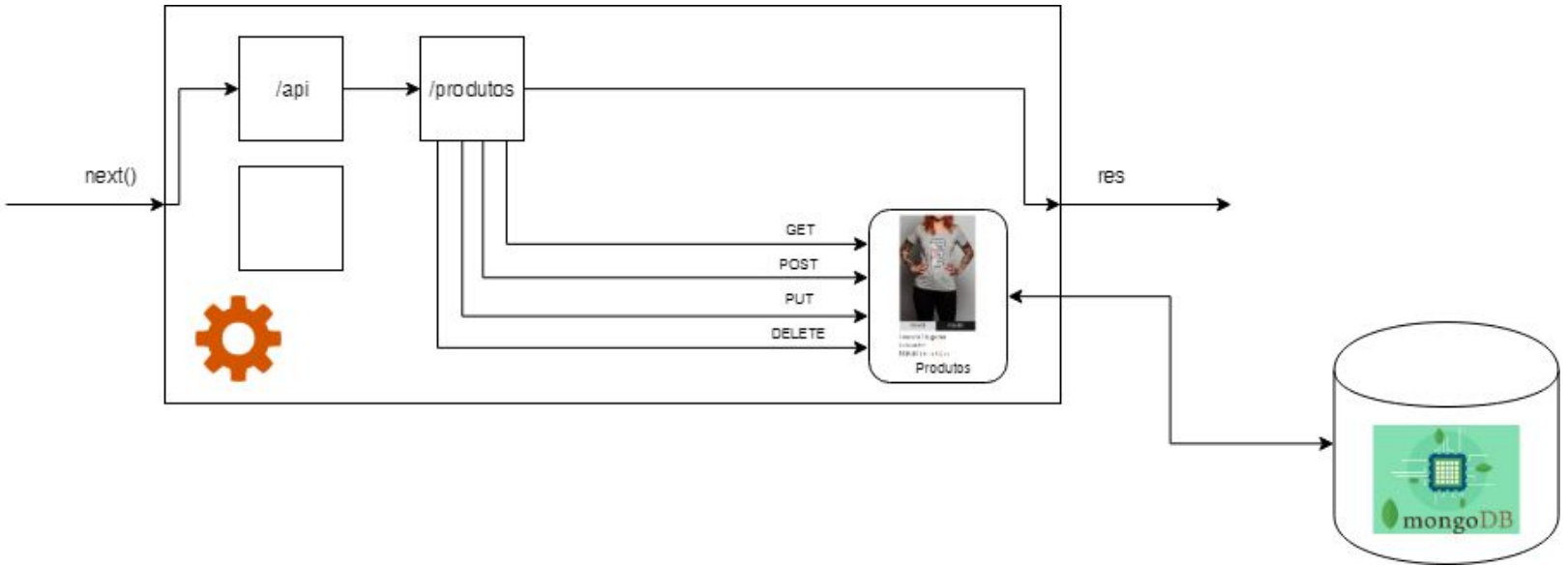
routes.js

```
const openApi = express.Router()
server.use('/api', openApi) //Obriga que todas as rotas em openApi usem o prefixo /api
```



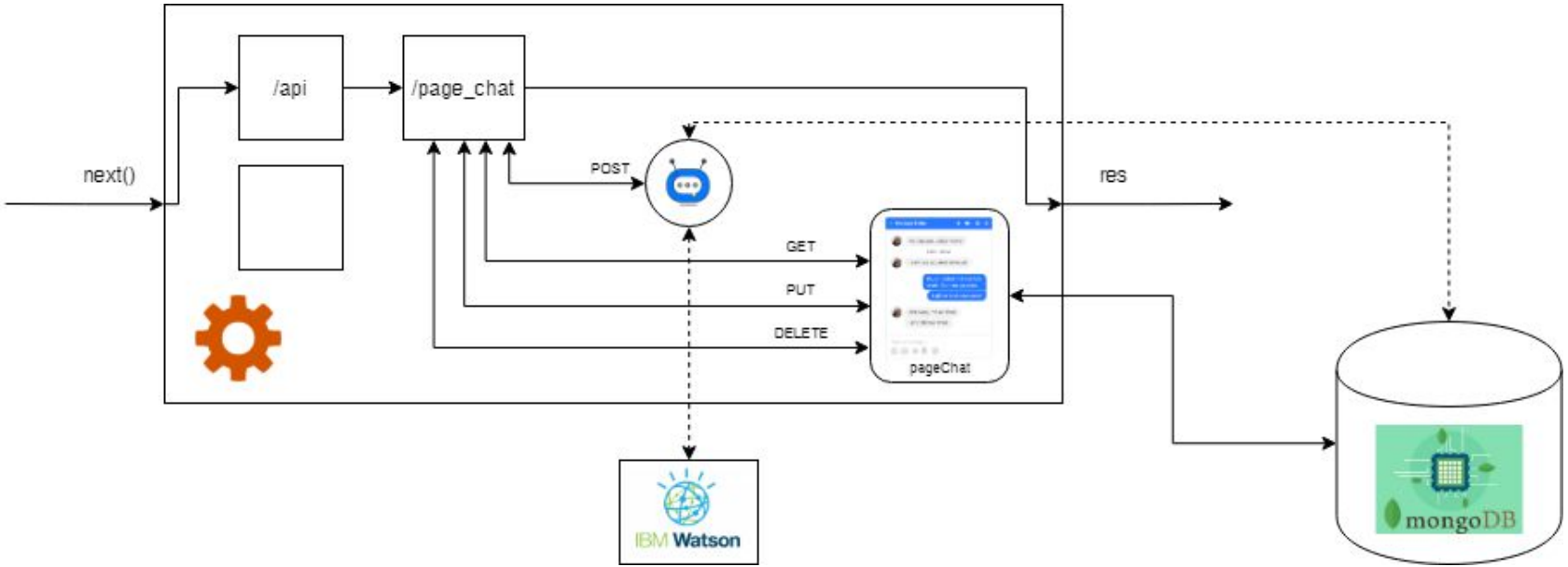
routes.js

```
ProdutosService.register(openApi, '/produtos')
```



routes.js

```
openApi.post('/page_chat', (req, res, next) => { // http://localhost:3003/api/page_chat (POST)
...
```



chatbot.js

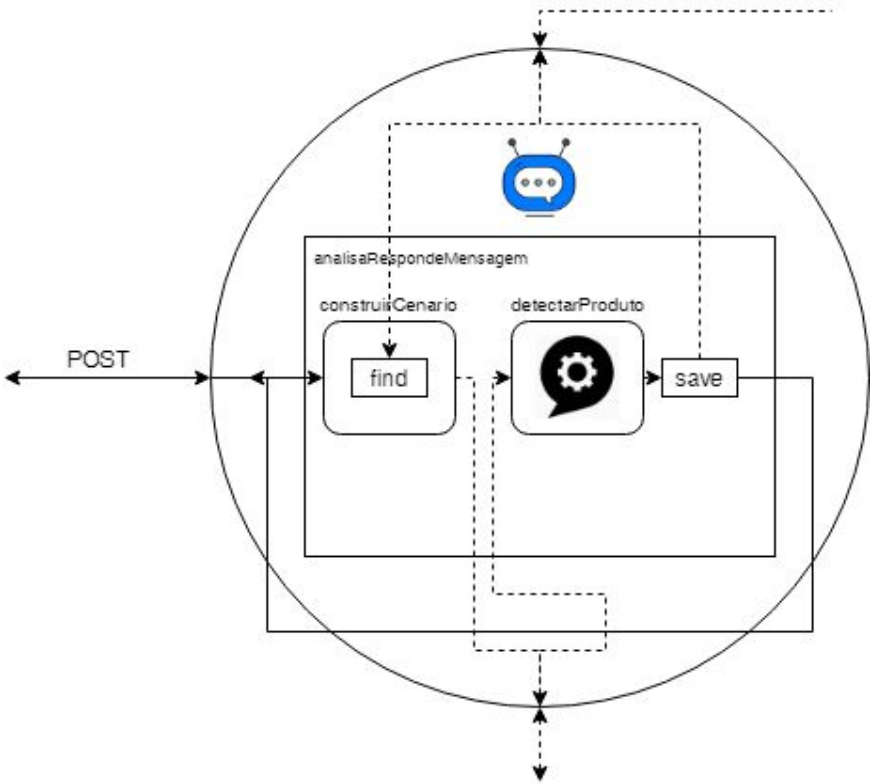
Dividido em 3 funções:

```
module.exports.analisarResponderMensagem // Função principal. Dentro dela são chamadas as outras duas.
```

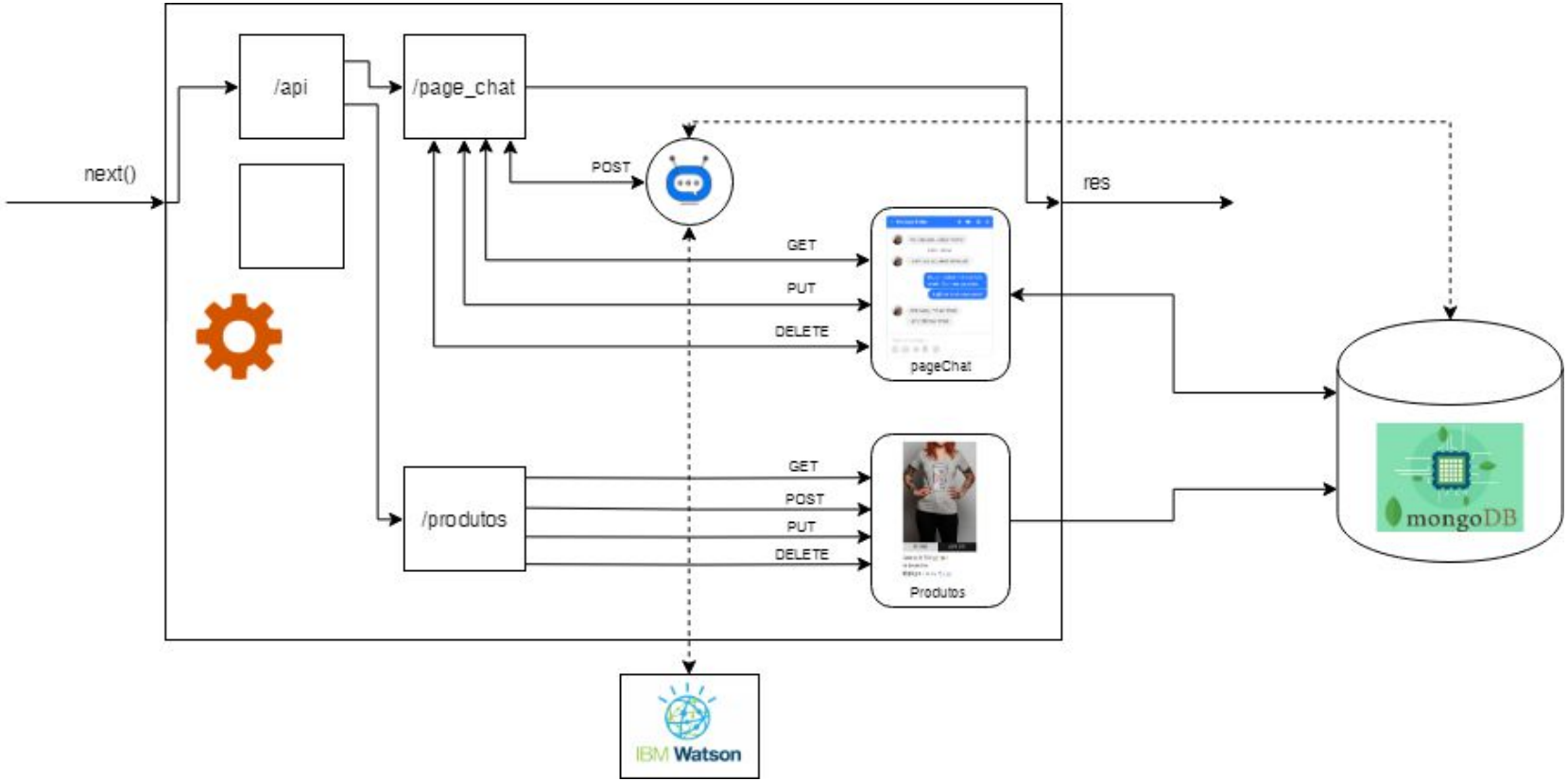
```
construirCenario // Essa função checa se o usuário possui registro no banco. Caso não exista, cria um. Se ja existe, somente o obtém. Por fim é retornado esse registro.
```

```
detectarProduto // Essa função pega todo o contexto obtido do watson, e faz as devidas alterações. Seja ela uma busca no banco de dados para alteração do contexto, ou mudanças radicais na resposta do Watson. São analisadas para as decisões as intenções, entidades e até mesmo os diálogos se você quiser.
```

chatbot.js


analisaRespondeMensagem
construirCenario
detectarProduto
POST
find
save

Final


next()
/api
/page_chat
POST
res
GET
PUT
DELETE
pageChat
/produtos
GET
POST
PUT
DELETE
Produtos
mongoDB
IBM Watson

Exercício Casa

Aula 05

Construir um serviço que possa atender aos exercícios anteriores (ou trabalho, se já estiver sendo executado.)