# Aviso legal



# Serviços para Aplicações Web e Móveis

## Framework Express

Fábio Marques (fabio@ua.pt)
Mestrado em Informática Aplicada

Escola Superior de Tecnologia e Gestão de Águeda
Universidade de Aveiro

- é uma framework minimal para o desenvolvimento de aplicações Web e Mobile com o Node.js
  - Permite criar aplicações de uma página, multipágina, híbridas, mobile e Web
  - Permite o desenvolvimento de funcionalidades backend para aplicações Web e e desenvolvimento de API
- JavaScript
- Multiplataforma
- Suporta a arquitetura MVC

- Routing: tem um sistema de routing simples e flexivel, permitindo definir rotas para os diferentes pedidos HTTP e associar funções que são executadas quando uma determinada rota é acedida;
- Middleware: permite o acesso ao pedido, à resposta e à função seguinte no ciclo de vida do pedido. As funções middleware podem executar código, fazer alterações ao pedido e à resposta, terminar o ciclo de vida do pedido ou chamar a função middleware seguinte;
- Templating: suporta a utilização de motores de template para a renderização de páginas HTML (pug, ejs, handlebars, etc) (não abordado nesta unidade curricular);

- Gestão de erros: tem um sistema de gestão de erros que permite a definição de funções para o tratamento de erros;
- RESTful API: a sua simplicidade e flexibilidade permite a criação de endpoints e o manuseamento de pedidos HTTP de uma forma fácil;
- Integração de bibliotecas de terceiros: permite a integração de bibliotecas e módulos de terceiros para estender as suas funcionalidades.

- Minimalista: foi desenhado para ser minimalista e flexível, permitindo a criação de aplicações Web e Mobile de forma rápida e eficiente;
- rápido e eficiente: tem um sistema de routing e middleware que permite a execução de código de forma rápida e eficiente;
- Routing robusto: tem um sistema de routing que permite a definição de rotas para os diferentes pedidos HTTP;
- Suporte para middleware: tem um sistema de middleware que permite a execução de código antes, durante e depois do ciclo de vida do pedido. O middleware pode ser aplicado globalmente ou apenas a rotas específicas;

- Criado para API: dada a sua simplicidade e facilidade de definição de rotas é uma opção adequada para a criação de API;
- De fácil aprendizagem: a curva de aprendizagem da framework Express.js é bastante reduzida, sendo uma boa opção para quem está a começar a desenvolver aplicações Web e Mobile com o Node.js.

- A ferramenta express-generator permite gerar o esqueleto de uma aplicação Express.

```
npx express-generator
```

**Criar e executar o projeto:**

1. Criar a pasta a conter o projeto
2. Executar o express-generator
3. Instalar as dependências (npm install)
4. Executar a aplicação (npm start)
5. Abrir o browser e aceder ao endereço `http://localhost:3000`

Explore a estrutura de pastas e ficheiros criados.

A Framework Express é composta por um conjunto de objetos e funções.

A função `express()` cria uma aplicação Express, sendo exportada pelo módulo `express` (`https://expressjs.com/en/4x/api.html#express`)

**Objetos**

- Express Application ('app'): é uma instância da aplicação Express. É responsável por configurar middleware, rotas e tratar dos pedidos e respostas HTTP.
- Request ('req'): representa o pedido HTTP. Contém informação sobre o pedido, como os cabeçalhos, o corpo do pedido, os parâmetros, etc.

- Response ('res'): representa a resposta HTTP. Contém métodos para definir o código de estado, cabeçalhos, corpo e enviar a resposta ao cliente.
- Next Function ('next'): esta função é utilizada nas funções middleware para passar o controlo para a função middleware seguinte.
- Router ('router'): é utilizado para criar instâncias de rotas. Permite a definição de rotas e a associação de funções middleware a essas rotas. Podem ser utilizadas para criar módulos de rotas, permitindo uma melhor organização do código.

- Tem métodos para o reencaminhamento de pedidos, configuração de middleware, registo de motores de template, etc.
- Tem um conjunto de propriedades que permite definir o comportamento da aplicação.

```
app.METHOD(PATH, HANDLER) \\ reencaminhamento de pedidos

\\ app: instancia da aplicação express.
\\ METHOD método HTTP, em minusculas (get, post, put, patch, ...).
\\ PATH string, pattern ou expressão regular correspondendo a caminho
    no servidor
\\ HANDLER função(oes) a executar quando a rota coincide com o método e
     o caminho.
```

Mais informação em `https://expressjs.com/en/4x/api.html#app`

Representa um pedido HTTP

| Método | Descrição |
|---|---|
| req.accepts() | Verifica se o conteúdo é aceitável |
| req.get() | Devolve o valor correspondente ao campo indicado |
| req.is() | Devolve o tipo de conteúdo associado ao campo indicado |
| req.param() | Devolve o valor do parametro indicado |

**Fonte:** `https://expressjs.com/en/4x/api.html#req`

Representa uma resposta HTTP

| Método | Descrição |
|---|---|
| res.download() | Permite o download do ficheiro |
| res.end() | Termina o processo de resposta |
| res.json() | Envia uma resposta em json |
| res.jsonp() | Envia uma resposta em json com suporte para jsonp |
| res.redirect() | Redireciona um pedido |
| res.render() | Renderiza um template |
| res.send() | envia uma resposta de vários formatos |
| res.sendFile() | Envia um ficheiro como um octeto |
| res.sendStatus() | Define o estado da resposta e envia-o |

**Fonte:** `https://expressjs.com/en/4x/api.html#res`

```
var router = express.Router(); // cria um novo objeto router

// Coincide com todos os métodos HTTP
router.all(path, [callback,...]callback)
// METHOD é um dos métodos HTTP
router.METHOD(path, [callback, ...] callback)
// Adiciona callbacks para parametros na rota
router.param(name, callback)
```

**Fonte:** https://expressjs.com/en/4x/api.html#router

Em combinação com os métodos utilizados no pedido HTTP, definem endpoints de uma API.
Para criar estes caminhos podem-se utilizar strings, padrões de string ou expressões regulares.

Exemplos:

```
path = '/';
path = '/help';
path = '/ab?cd';
path = '/ab+cd';
path = '/ab(cd)?e';
...
```

- são utilizados para extrair valores do URL
- são definidos utilizando ':' seguindo-se o nome do parâmetro
- são úteis quando se pretende aceder a um recurso específico

Exemplo:

```
path = "/users/:userId/books/:bookId"
```

- são incluídos no URL após o símbolo '?' e são pares chave-valor separados por '&', mas não estão diretamente relacionados com a estrutura do URL
- são utilizados para enviar mais informação para o servidor, na maioria dos casos para filtrar os resultados, ordenar, etc.

Exemplo:

```
const name = req.query.name
```

Comportam-se como um middleware para satisfazer o pedido.
Podem ser na forma de uma função, de um array de funções ou ambos.

```
app.get('/example/a', function (req, res) { ... })

app.get('/example/b',
function (req, res, next) { ...
  next()
},
function (req, res) { ... }
)
```

- Criar um projeto Express.js para a gestão de livros da sua biblioteca pessoal utilizando o express-generator
- Crie um array de objetos para representar os livros da sua biblioteca (este array vai ser utilizado para simular uma base de dados)
- Implemente um conjunto de rotas para a gestão dos livros (listar, pesquisar (por author, por tipo, por título, etc.) adicionar, editar e remover)
- Teste a sua aplicação

- É crucial para gerir os erros que podem ocorrer durante a execução da aplicação. Erros comuns incluem: problemas de ligação à BD, erros de validação, erros de autenticação, etc.
- Middleware de erro
  - Express tem um mecanismo de tratamento de erros incorporado como middleware
  - As funções middleware de erro têm 4 argumentos (err, req, res, next)
  - quando ocorre um erro, o Express ignora o middleware normal e passa o controlo para o middleware de erro

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something went wrong!');
});
```

- Podem ser criadas classes de erro personalizadas para estruturar e tratar tipos específicos de erros.

```
class CustomError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
  }
}
```

- Podem ser lançados e capturados em rotas e middleware

```
app.get('/example', (req, res, next) => {
  try {
    // Some code that may throw an error
    throw new CustomError('Custom error message', 404);
  } catch (err) {
    next(err); // Pass the error to the next middleware
  }
});
```

- Cada função middleware tem acesso ao pedido, à resposta e à função seguinte no ciclo de vida do pedido, podendo alterar qualquer um destes objetos ou terminar o ciclo de vida do pedido.
- As funções de middleware são executadas pela ordem pela qual são definidas
- Exemplos de Middleware:
  - middleware de autenticação
  - middleware de análise do corpo do pedido
  - middleware de logging

- Criar funções de middleware

```
const customMiddleware = (req, res, next) => {
  // Middleware logic
  console.log('Custom middleware executed');
  next(); // Move to the next middleware in the stack
};
```

- Utilizar funções de middleware

```
app.use(customMiddleware);
```

- Middleware com parâmetros

```
const paramMiddleware = (param) => (req, res, next) => {
  // Middleware logic using the parameter
  console.log(‘Middleware with parameter: param‘);
  next();
};
```

- Tendo em consideração o projeto criado anteriormente, personalize o tratamento de erros na aplicação a rotas específicas.
- Crie uma função de middleware que registe o tempo de execução de cada pedido e aplique-a a todas as rotas.
- Crie uma função de middleware que faça o registo de pedidos e aplique-a aos pedidos de adição e remoção de livros.