Resumo Sails.js

Sails.js Cheat Sheet

Ordem (*)	Comando	Finalidade	Observação
1.	#> npm install sails -g	Instalar o *framework* na estação de trabalho, para acesso global	Requer o Node.js 16+ e Npm 8+ instalados
2.	#> sails new meu-prototipo	Gerar um novo Protótipo de aplicação *Full JavaScript* estruturado.	Pode gerar um Protótipo *Full Stack* (opção 1) ou somente *Backend* (opção 2)
3.	#> sails lift	Executar o protótipo Sails em modo de desenvolvimento	Este comando força o uso da variável de ambiente NODE_ENV=development
4.	#> npm run deploy	Produz a versão de distribuição do Protótipo Sails	a) Pré-configurada para Linux, logo necessita ser ajustada para Windows. b) Considera que o desenvolvedor possui uma conta Heroku. c) Vide arquivo 'package.json'.
5.	#> sails generate model nome	Gera um arquivo de Modelo de Dados 'Nome.js' para uso pelo modelo **M**VC[1]	O *framework* Sails irá reconhecer e validar este arquivo como modelo de dados e objeto relacional durante a inicialização do Protótipo
6.	#> sails generate page nome	Gera uma página visão do usuário para uso pelo modelo M**V**C. Gera também uma rota para a página para uso pelo modelo MV**C**.	O comando irá produzir 3 arquivos: - views/nome.ejs; - assets/js/nome.page.js; - api/controllers/view-nome.js
7.	#> sails generate action nome	Gera um controle de ação em resposta a uma requisição para uso pelo modelo MV**C**	O comando irá produzir 1 arquivo seguindo o padrão Action 2 como: - api/controllers/nome.js
8.	#> sails generate controller nome	Gera um controle de ações de resposta a requisições, seguindo o modelo MV**C**	O comando irá produzir 1 arquivo seguindo o padrão Action ou Clássico como: - api/controllers/NomeController.js
9.	#> sails generate api nome	Gera uma RESTFul API para uso pelo modelo **M**V**C**	O comando irá produzir 2 arquivos: - api/controllers/NomeController.js - api/models/Nome.js

10.	#> sails generate script nome	Gera um *script file* que poderá ser executado posteriormente, tanto pela aplicação em tempo de execução quanto como comando shell	Para executar como comando shell: #> sails run nome Para executar como parte de código, em tempo de execução: `await require('<path>/scripts/nome').fn()`
11.	#> sails run nome	Executa *scripts* criados pelo Sails e que estão localizados no diretório ‘./scripts’ do Protótipo	É obrigatória a existência de uma função fn() no corpo do código JavaScript do arquivo ‘./scripts/nome.js’
12.	#> sails generate helper nome	Gera um componente *Helper* que pode ser reutilizado por toda a aplicação.	Um componente *Helper* é validado e carregado em memória durante a inicialização do Protótipo. Pode ser utilizado em tempo de execução com o comando: `await sails.helpers.nome(params)`

13.	#> sails console	Inicia um console gerenciável do Sails.
		
		sails>
		
		Para sair, utilize:
		
		sails> .exit
			Este console permite executar funções internas diretamente. Exemplo:
			`> await User.find()`
			
			`> await sails.helpers.nome(params)`
			
			`> sails.config.security.csrf`
			
			`> process.env`

14.	#> npm run test	Executa testes de qualidade do código (lint) e unários (custom-tests)	d) Pré-configurada para Linux, logo necessita ser ajustada para Windows. e) Considera que o desenvolvedor escreveu seus próprios testes. Vide arquivo ‘package.json’.

(*) Ordem em que os comandos vão sendo necessários, naturalmente. Claro que não é uma regra!
(1) MVC – Model View Controll – Modelo Visão Controle: técnica de integração entre dados e telas por meio de controles por rota de requisições HTTP.

Arquivos Relevantes Sails.js	
Local/nome	Função
api/policies/is-logged-in.js	Contém algoritmo de teste de segurança de acesso (está logado?)
assets/js/cloud.setup.js	Contém rotas de ações para automação pelo componente Cloud.js
config/routes.js	Contém as rotas de ações e controles associados a cada rota
config/policies.js	Contém regras de validação para controle de acesso às rotas
config/local.js	Sobrescreve todas as definições dos outros arquivos em ./config/* e ./env/*