

Antony Kervazo-Canut

Yeoman for Teenagers

**TEMPLATES DE PROJETS
COMPLETS**



SOMMAIRE

Introduction	3
Installation de Yeoman	4
Les générateurs	5
Création d'un générateur	6
Structure d'un générateur	7
Premier fichier	8
Les prompts	9
Les templates	10
Syntaxe EJS	12
Templates EJS	13
Types de Prompting	15
install()	17
Sous-Générateurs	18
Tests de générateurs	19

Introduction



Yeoman est un outil de génération de squelette d'applications (application scaffolding tool) qui aide les développeurs à démarrer rapidement de nouveaux projets. Il propose un écosystème de générateurs pour de nombreux types d'applications et de frameworks, allant du front-end au back-end, en passant par les applications complètes.

L'utilisation de Yeoman permet de :

- **Gagner du temps** : Yeoman automatise le processus fastidieux de mise en place de nouveaux projets, en configurant l'environnement de développement avec les outils et les configurations nécessaires.
- **Promouvoir les bonnes pratiques** : Les générateurs sont souvent conçus par des experts de la communauté, incorporant les meilleures pratiques et les dernières conventions.
- **Faciliter la cohérence** : En utilisant Yeoman au sein d'une équipe, vous assurez que chaque membre démarre avec la même structure de projet et les mêmes outils, réduisant ainsi les incohérences.

Installation de Yeoman



Avant d'installer Yeoman, vous devez avoir Node.js et npm (le gestionnaire de paquets de Node) installés sur votre machine.

```
● ● ●  
# Vérifier si Node.js et npm sont déjà installés  
node -v  
npm -v  
  
# Installer Node.js et npm  
# Visitez https://nodejs.org/ pour télécharger et installer  
la version recommandée pour votre système d'exploitation.  
  
# Installer Yeoman globalement avec npm  
npm install -g yo
```

Les générateurs



Un générateur Yeoman est un plugin qui fournit un ensemble de instructions et de templates pour initialiser un nouveau projet. Il peut créer des fichiers, installer des dépendances et configurer automatiquement un projet selon les technologies choisies.

Pour trouver un générateur qui correspond à vos besoins, vous pouvez utiliser le site web officiel de Yeoman ou npm ou bien le faire en CLI.

Liste des générateurs

```
● ● ●

# Rechercher un générateur via npm
npm search yeoman-generator [mot-clé]

# Installer un générateur spécifique
npm install -g generator-[nom]

# Initialiser un projet avec un générateur spécifique
yo [nom-du-générateur]
```

Lorsque vous exécutez la dernière commande, le générateur vous posera une série de questions pour personnaliser votre projet. Ces questions peuvent inclure le nom du projet, les fonctionnalités à inclure, et les options de configuration.

Création d'un générateur



Créer votre propre générateur Yeoman est un excellent moyen de personnaliser vos projets et de standardiser les configurations au sein de votre équipe. Cela implique de développer un module npm qui interagit avec l'API Yeoman pour générer des fichiers, installer des dépendances, et configurer des projets selon vos spécifications.



```
# Créer un dossier pour le générateur
mkdir generator-monprojet
cd generator-monprojet

# Initialiser le dossier comme un module npm
npm init

# Installer yeoman-generator comme une dépendance
npm install yeoman-generator

# Créer le dossier des générateurs et le sous-dossier `app`
mkdir -p generators/app

# Créer le dossier des templates
mkdir generators/app/templates
```

Structure d'un générateur



La structure typique d'un générateur Yeoman comprend un dossier `generators`, dans lequel vous placerez tous les sous-générateurs. Le sous-générateur le plus courant est appelé `app`, mais vous pouvez en créer d'autres selon les besoins de votre projet.

```
generator-monprojet/
├── package.json          # Dépendances et métadonnées.
├── .yo-rc.json           # Configuration Yeoman (optionnel)
└── README.md              # Documentation du générateur
   └── generators/         # Dossier des sous-générateurs.
      ├── app/               # Le sous-générateur principal
      │   ├── index.js        # Le fichier principal
      │   └── templates/       # Dossier contenant les modèles
      │       ├── _package.json
      │       ├── _README.md
      │       └── src/
      │           ├── index.html
      │           └── styles.css
      └── prompts.js          # (Optionnel, si séparé d'index.js)
   └── component/            # Un autre sous-générateur
      ├── index.js
      └── templates/
          └── component.js
```



Premier fichier

Le cœur de chaque sous-générateur est le fichier `index.js`, qui définit son comportement. Dans le dossier `generators/app`, créez un fichier `index.js`. Ce fichier exportera une classe étendant `Generator` de Yeoman, où vous définirez plusieurs méthodes correspondant aux différentes étapes de la génération d'un projet.

```
● ● ●

// Importer la dépendance de Yeoman Generator
const Generator = require('yeoman-generator');

// Définition de la classe du générateur
module.exports = class extends Generator {
    // Méthode d'initialisation
    initializing() {
        this.log('Initialisation de votre projet ... ');
    }

    // Méthode pour interroger l'utilisateur
    prompting() {
        // Exemple : demander le nom du projet
    }

    // Méthode pour écrire des fichiers
    writing() {
        // Exemple : copier des templates
    }

    // Méthode pour installer des dépendances
    install() {
        // Exemple : exécuter `npm install`
    }
};
```

Les prompts



L'un des aspects les plus puissants de la création d'un générateur Yeoman est la capacité d'interagir avec l'utilisateur pour personnaliser le projet généré en fonction de ses entrées.

La méthode `prompting()` dans votre générateur est l'endroit où vous définirez les questions à poser à l'utilisateur. Utilisez l'API `this.prompt()` pour poser des questions, qui retourne une promesse avec les réponses.

```
● ● ●

prompting() {
  return this.prompt([
    {
      type: 'input',
      name: ' projectName',
      message: 'Quel est le nom de votre projet ?',
      default: this.appname // Par défaut, utilisez le nom du
dossier actuel
    },
    {
      type: 'confirm',
      name: ' addFeatureX',
      message: 'Voulez-vous inclure la fonctionnalité X ?',
      default: true
    }
  ]).then(answers => {
    this.projectName = answers.projectName;
    this.addFeatureX = answers.addFeatureX;
  });
}
```

Les templates



Une fois que vous avez recueilli les réponses de l'utilisateur, utilisez ces informations dans la méthode writing() pour personnaliser la génération des fichiers.

Dans vos fichiers de templates (par exemple, index.html.tpl), utilisez des placeholders pour les parties du fichier que vous souhaitez personnaliser en fonction des réponses de l'utilisateur. Yeoman remplacera ces placeholders par les valeurs correspondantes lors de la génération du fichier.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title><%= projectName %></title>
  </head>
  <body>
    <h1>Bienvenue dans <%= projectName %>!</h1>
    <% if (addFeatureX) { %>
      <p>La fonctionnalité X est incluse!</p>
    <% } %>
  </body>
</html>
```

Les templates



```
● ○ ● ●  
  
writing() {  
  this.fs.copyTpl(  
    this.templatePath('index.html.tpl'),  
    this.destinationPath('index.html'),  
    { projectName: this.projectName }  
  );  
  
  if (this.addFeatureX) {  
    // Logique pour inclure la fonctionnalité X  
  }  
}
```

En combinant des questions dynamiques avec la personnalisation des templates, vous pouvez créer des générateurs Yeoman qui s'adaptent aux besoins spécifiques de l'utilisateur, rendant chaque projet généré unique. Lors de la génération des fichiers, Yeoman vérifie si un fichier existe déjà et, par défaut, vous demande si vous souhaitez l'écraser. Vous pouvez personnaliser ce comportement en utilisant l'option `conflicter`.

```
● ○ ● ●  
  
this.conflicter.force = true; // Écrase les fichiers sans demander
```

Syntaxe EJS



La gestion avancée des templates avec EJS (Embedded JavaScript) dans Yeoman permet une grande flexibilité dans la génération de fichiers. EJS utilise du JavaScript intégré dans des fichiers de template, ce qui vous permet d'ajouter de la logique conditionnelle, des boucles, et de manipuler des données de manière dynamique lors de la création de fichiers. Voici une exploration plus détaillée de ce que vous pouvez faire avec EJS dans vos templates Yeoman.

```
● ● ●

writing() {
  const templateData = {
    pageTitle: 'Ma Page',
    heading: 'Bienvenue sur Ma Page',
    showList: true,
    listItems: ['Élément 1', 'Élément 2', 'Élément 3']
  };

  this.fs.copyTpl(
    this.templatePath('index.html.ejs'), // Chemin du
    template source
    this.destinationPath('index.html'), // Destination
    du fichier généré
    templateData // Données pour
    remplacer les variables du template
  );
}
```

Templates EJS



```
// Variables
Bienvenue, <%= userName %>!

// Conditions
<% if (isAdmin) { %>
    <p>Vous avez accès à la section administrative.</p>
<% } else { %>
    <p>Vous êtes un utilisateur standard.</p>
<% } %>

// Boucles
<% for(let i = 0; i < users.length; i++) { %>
    <p><%= users[i].name %></p>
<% } %>
// —
<% for(let user of users) { %>
    <p><%= user.name %></p>
<% } %>

<% for(let key in userDetails) { %>
    <p><%= key %>: <%= userDetails[key] %></p>
<% } %>
// —
<%
let i = 0;
while(i < users.length) { %>
    <p><%= users[i].name %></p>
<% i++;
} %>
// —
<%
let i = 0;
do { %>
    <p><%= users[i].name %></p>
<% i++;
} while (i < users.length); %>
```

Templates EJS



```
● ● ●  
  
// Fonctions Javascript  
<% const date = new Date(); %>  
<p>Nous sommes le <%= date.toDateString() %>.</p>  
  
// Commentaires  
<%# Ceci est un commentaire et ne sera pas affiché dans le  
fichier généré. %>  
  
// EJS échape le HTML par défaut (pour prévenir les  
injections XSS)  
<p><%= userContent %></p> ←— Échappe le HTML —>  
<p><%- userContent %></p> ←— N'échappe pas le HTML —>
```

Vous pouvez même templatiser des templates pour les réutiliser plusieurs fois. Cela donne les sous-templates.

```
● ● ●  
  
<%- include('header.ejs') %>  
<p>Contenu principal ici ... </p>  
<%- include('footer.ejs') %>
```

Types de Prompting



```
{ // Utilisé pour recevoir une chaîne de caractères simple
  type: 'input',
  name: 'name',
  message: 'Quel est votre nom ?',
  default: 'Mon Nom' // Valeur par défaut
}

{ // Poser une question oui/non
  type: 'confirm',
  name: 'someFeature',
  message: 'Voulez-vous activer cette fonctionnalité ?',
  default: true // Valeur par défaut
}

{ // Liste de choix, une seule option possible
  type: 'list',
  name: 'script',
  message: 'Quel script souhaitez-vous ajouter ?',
  choices: ['React', 'Vue', 'Angular']
}

{ // Liste de choix indexés, une seule option possible
  type: 'rawlist',
  name: 'size',
  message: 'Quelle taille ?',
  choices: ['Petit', 'Moyen', 'Grand']
}

{ // Liste de choix, plusieurs options possibles
  type: 'checkbox',
  name: 'components',
  message: 'Quels composants souhaitez-vous inclure ?',
  choices: ['Header', 'Footer', 'Sidebar']
}
```

Types de Prompting



```
{ // Input mais l'entrée dans le terminal est masquée
  type: 'password',
  name: 'userPassword',
  message: 'Entrez un mot de passe',
  mask: '*' // Masque l'entrée avec des étoiles
}

{ // Liste de choix avec touches de sélection rapide.
  type: 'expand',
  name: 'conflict',
  message: 'Que voulez-vous faire en cas de conflit de
fichiers ?',
  choices: [
    {
      key: 'y',
      name: 'Écraser',
      value: 'overwrite'
    },
    {
      key: 'a',
      name: 'Écraser ce fichier et tous les suivants',
      value: 'overwrite_all'
    },
    {
      key: 'x',
      name: 'Sortir',
      value: 'exit'
    }
  ]
}
```

install()



La méthode `install()` dans votre générateur est utilisée pour exécuter les commandes d'installation des dépendances. Yeoman peut gérer l'installation via `npm`, `bower`, ou `yarn`, selon la configuration de votre générateur et les préférences du projet.

```
install() {  
  // Installer les dépendances npm  
  this.npmInstall();  
  
  // Vous pouvez également spécifier des paquets à installer  
  this.npmInstall(['lodash', 'jquery'], { 'save-dev': true  
});  
  
  // Pour utiliser yarn à la place de npm  
  this.yarnInstall(['lodash', 'jquery'], { 'dev': true });  
}
```

Sous-Générateurs



Créez un dossier `sous generators/` pour chaque sous-générateur, par exemple `generators/component/`, avec son propre `index.js`.

```
● ● ●

// generators/component/index.js
const Generator = require('yeoman-generator');

module.exports = class extends Generator {
  prompting() {
    // Demandez le nom du composant, etc.
  }
  writing() {
    // Créez le fichier du composant
  }
};
```

Dépuis l'`index.js` du générateur principale, vous pouvez invoquer le sous-générateur.

```
this.composeWith('monprojet:component', { args:
  [this.theName] });
```

Tests de générateurs



La mise en place de tests solides est essentielle pour assurer la fiabilité et la maintenabilité de votre générateur Yeoman au fil du temps. Les tests avancés peuvent couvrir non seulement les aspects fondamentaux comme la création de fichiers mais aussi les interactions utilisateur, la logique conditionnelle, et l'intégration avec des services externes.

```
● ○ ●

const helpers = require('yeoman-test');
const assert = require('yeoman-assert');

describe('Mon Générateur', () => {
  beforeAll(() => {
    return helpers.run(path.join(__dirname,
      '../path/to/your/generator'))
      .withPrompts({someAnswer: true}) // Simuler des
    réponses utilisateur
      .toPromise();
  });

  it('crée les fichiers de base du projet', () => {
    assert.file(['package.json', 'index.js']);
  });

  it('remplit le package.json avec le bon nom de projet', () => {
    assert.jsonFileContent('package.json', {
      name: 'mon-projet'
    });
  });
});
```

Dans la même collection

Orchestration et Gestion de Conteneurs



Infrastructure as Code



Sécurité & Gestion des secrets



Développement & CI/CD



↓ FOLLOW ME ↓



[ANTONYCANUT](#)



[ANTONY KERVAZO-CANUT](#)