



# أساسيات إطار العمل Vue.js

حسام برهان

# أساسيات إطار العمل Vue.js

مقدمة إلى بناء واجهات المواقع وتطبيقات الويب عبر Vue.js

تأليف

حسام برهان

تحرير وإشراف

جميل بيلوني

جميع الحقوق محفوظة © 2021 أكاديمية حسوب

النسخة الأولى v1.0

هذا العمل مرخص بموجب رخصة المشاع الإبداعي: نسب المصنف - غير تجاري - الترخيص

بالمثل 4.0 دولي



## عن الناشر

أنتج هذا الكتاب برعاية شركة حسوب وأكاديمية حسوب.



تهدف أكاديمية حسوب إلى توفير فصول وكتب عالية الجودة في مختلف المجالات وتقديم دورات شاملة لتعلم البرمجة بأحدث تقنياتها معتمدةً على التطبيق العملي الذي يؤهل الطالب لدخول سوق العمل بثقة.



حسوب مجموعة تقنية في مهمة لتطوير العالم العربي. تبني حسوب منتجات تركز على تحسين مستقبل العمل، والتعليم، والتواصل. تدير حسوب أكبر منصتي عمل حر في العالم العربي، مستقل وخمسات ويعمل في فيها فريق شاب وشغوف من مختلف الدول العربية.

# جدول المحتويات

9

تقديم

11

## 1. مدخل إلى Vue.js

12

1.1 ما هو إطار العمل framework في JavaScript؟

12

1.2 تطبيقك الأول مع Vue.js

15

1.3 إضافة مزايا جديدة لتطبيقنا الأول

17

1.4 العمل على حاسوب محلي

18

1.5 ختام الفصل

18

1.6 تمارين داعمة

21

## 2. التعامل مع DOM

22

2.1 فهم قوالب Vue.js

22

2.2 الوصول إلى البيانات والتوابع من كائنات Vue.js

23

2.3 الربط مع السمات Attributes

25

2.4 كتابة شيفرة HTML خام

26

2.5 التعامل مع الأحداث (Events)

31

2.6 استخدام الربط ثنائي الاتجاه

32

2.7 ختام الفصل

32

2.8 تمارين داعمة

34

## 3. الموجهات الشرطية والتكرارية

35

3.1 كتابة شفرات JavaScript مباشرة ضمن القوالب

36

3.2 التصيير الشرطي باستخدام v-if و v-else و v-else-if

38

3.3 الفرق بين v-if و v-show

40

3.4 تصيير القوائم باستخدام v-for

41	3.5	المرور على خصائص كائن
44	3.6	ختم الفصل
44	3.7	تمارين داعمة

## 4. المزيد حول كائن Vue.js 45

46	4.1	إنشاء أكثر من كائن Vue.js
48	4.2	الوصول إلى عناصر HTML مباشرة
49	4.3	الخصائص المحسوبة في Vue.js
51	4.4	الخصائص المراقبة ضمن كائن Vue.js
53	4.5	تثبيت قالب جديد باستخدام \$mount()
54	4.6	فصل القالب عن عنصر HTML المستهدف
56	4.7	ما هو المكون (Component)؟
59	4.8	ختم الفصل
59	4.9	تمارين داعمة

## 5. مدخل إلى التعامل مع المكونات 60

61	5.1	تجهيز هيكل التطبيق على حاسوب محلي
65	5.2	بناء مكون جديد: مكون المهام
70	5.3	تحسين تجربة الاستخدام للمكون
71	5.4	تمرير وسائط إلى المكونات
73	5.5	إنشاء أكثر من نسخة من المكون ضمن نفس الصفحة
76	5.6	إضافة ميزة الترشيح لمكون المهام
78	5.7	إضافة ميزة مهمة جديدة لمكون المهام
81	5.8	ختم الفصل
81	5.9	تمارين داعمة

## 6. التعمق في مكونات Vue.js 82

83	6.1	بناء تطبيق نموذجي (مشروبات حسوب)
88	6.2	إضافة وسائل تنقيح متطورة لتطبيقات Vue.js
89	6.3	المكونات المتداخلة
91	6.4	تسجيل المكونات محليًا وتسجيلها على المستوى العام
93	6.5	تحديد المكون الذي اختاره المستخدم
96	6.6	التخاطب بين المكونات باستخدام أحداث مخصصة
100	6.7	ختم الفصل
<b>101</b>	<b>7.</b>	<b>إنشاء مشاريع باستخدام Vue CLI</b>
102	7.1	لماذا Vue CLI؟
102	7.2	إعداد Vue CLI
104	7.3	إنشاء مشروع جديد باستخدام Vue CLI
107	7.4	نظرة عامة على هيكل المشروع
117	7.5	ختم الفصل
<b>118</b>	<b>8.</b>	<b>التعامل مع دخل المستخدم عن طريق نماذج الإدخال</b>
119	8.1	بناء هيكل تطبيق بسيط
124	8.2	استخدام إطار العمل Bootstrap
125	8.3	استخدام بنى معطيات متقدمة مع عناصر الإدخال النصية
126	8.4	المعدلات (Modifiers) في Vue.js
127	8.5	التعامل مع مربعات الاختيار وأزرار الانتقاء
128	8.6	التعامل مع القائمة المنسدلة
129	8.7	إرسال البيانات
130	8.8	ختم الفصل
<b>131</b>	<b>9.</b>	<b>المرشحات Filter والمخاليط Mixin</b>
132	9.1	المرشح (Filter)

134	9.2 المخلوط (Mixin)
139	9.3 ختام الفصل
<b>140</b>	<b>10. استخدام Vue.js للاتصال بالإنترنت</b>
141	10.1 إنشاء قاعدة بيانات على Google Firebase
145	10.2 تثبيت مكتبة الاتصال بالإنترنت vue-resource
146	10.3 بناء تطبيق للقراءة والإضافة من وإلى قاعدة البيانات
155	10.4 إضافة ميزة تعديل البيانات للتطبيق السابق
162	10.5 ختام الفصل
<b>163</b>	<b>11. بناء تطبيقات ذات صفحة واحدة</b>
164	11.1 بناء هيكل التطبيق والتعرف على أجزائه الرئيسية
165	11.2 تحويل تطبيق إلى تطبيق SPA
182	11.3 ختام الفصل
<b>183</b>	<b>12. نشر تطبيق Vue.js إلى الإنترنت</b>
183	12.1 تجهيز التطبيق قبل النشر
184	12.2 نشر التطبيق على منصة Netlify
189	12.3 ختام الفصل



# تقديم

بدأت حكاية هذا الكتاب بدايةً تختلف عن غيره من الكتب التي عملنا عليها، إذ كنت أؤخر كتابًا عن إطار العمل Vue.js من المراجع الأجنبية الحرة أو مفتوحة المصدر لترجمته ف وقعت يدي على كتيب باسم The Vue Handbook لمؤلفه Flavio Copes ولكنه كتابًا مختصرًا صغيرًا لا يلم حتى بالأساسيات وإنما هو تقديم أو تعريف مختصر بإطار العمل Vue.js فاقترح حسام برهان -مؤلف هذا الكتاب- العمل على تأليف كتاب كامل يشمل أساسيات Vue.js ويكون مقدمة لمن يريد تعلم هذا الإطار، وهنا بدأت الحكاية وكتبنا معًا فهرس الكتاب الذي بين يديك الآن وشرعنا بالعمل.

سبب اختيارنا شرح إطار العمل Vue.js هو أنه إطار شامل واعد سريع الأداء ذاع الصيت يستعمل لبناء تطبيقات ويب، توجهت أنظارنا لإضافة محتوى عربي مميز يشمله بعد أن اخترنا كتابًا لتغطية مكتبة React الشهيرة مسبقًا وكتابًا حول Angular في خطة لإضافة مصادر تعليمية عالية الجودة تشرح أشهر أطر لغة جافاسكربت لبناء الواجهات الأمامية للمواقع وتطبيقات الويب.

حرصنا في هذا الكتاب أن يكون سهل الفهم والتطبيق يشمل أساسيات Vue.js التي ستتمكنك من بدء العمل سريعًا مع هذا الإطار متبعين قاعدة 20/80 (التي ننتهجها عادةً في كتبنا والتي تقول تعلم 20% من تلك المفاهيم التي تستعمل في 80% من حالات الاستخدام). اتبعنا في الكتاب النهج العملي التطبيقي إذ ستبني خلال فصول الكتاب عدة تطبيقات عملية وستتعلم كيفية نشرها وإطلاقها على الويب، واعتمدنا أيضًا الأسلوب السهل البسيط المختصر فخير الكلام ما قل ودل مع إضافة تمارين وأفكار في نهاية كل فصل ما أمكن لتطبيقها بنفسك.

حتى تستفيد أكبر استفادة من الكتاب، يجب أن تملك معرفة أساسية مسبقة بلغة HTML ولغة CSS ومعرفة جيدة بلغة جافاسكربت، إذ يركز الكتاب على إطار العمل Vue.js الذي هو أحد أطر عمل جافاسكربت ولن يتطرق الكتاب إلى شرح لغة جافاسكربت، وقد تجد صعوبة في فهم أجزاء الشيفرات والاستفادة من الكتاب دون تلك المعرفة المسبقة. انظر قسم [الكتب البرمجية](#) في أكاديمية حسوب لتنزيل مصادر لتلك اللغات.

يبدأ الكتاب فصوله الأولى بشرح مفهوم إطار العمل والتعرف على Vue.js وأخذك سريعًا وباختصار لبدء استعماله ببناء أول تطبيق عبره. ستبدأ من الفصل الثاني في Vue.js بالتعرف على إطار العمل Vue.js عن قرب وعن آلية عمله وكيفية استعماله في بناء الواجهات الأمامية ثم تنتقل بعدها إلى التعرف على الموجهات الشرطية والتكرارية واستعمالها في قوالب الواجهة الأمامية، يليها الدخول إلى عالم مكونات Vue.js وتعلم كيفية بناءها. ستبدأ بعدها في الفصل السابع الأمور المتقدمة إذ ستتعلم التعامل مع سطر أوامر Vue.js لبناء المشاريع والتطبيقات بسهولة ويسر ثم ستتعرف بعد ذلك على مفاهيم متقدمة في Vue.js وكيفية بناء تطبيقات ذات صفحة واحدة. ستتعلم في الفصول الأخيرة من الكتاب كيفية ربط تطبيقك بالإنترنت وإضافة قاعدة بيانات له وإطلاق إلى العالم الخارجي متحرين استعمال أشهر منصات النشر وأيسرها تعاملًا.

نرجو أن نكون قد وفقنا في إضافة مرجع عربي قيّم حول Vue.js يفيدك في تعلم وبدء استعمال ذلك الإطار لبناء تطبيقات ويب مفيدة، وندعو لك بالتوفيق والسداد في خطواتك القادمة وحياتك المهنية، وسنكون فخورين بما ستبنيه مما تعلمته من هذا الكتاب!

نسعد بالسماع منك لأي رأي أو تصحيح أو ملاحظة عبر بريد الأكاديمية [academy@hsoub.com](mailto:academy@hsoub.com)

جميل بيلوني

16/06/2021

# 1. مدخل إلى Vue.js

مرحبًا بك في هذا الكتاب التي سنتعلم فيه كيف سنتعامل مع إطار العمل Vue.js. هذا الإطار الواعد الذي يُسهّل العمل في تطوير الواجهة الأمامية للتطبيقات Frontend Applications إلى حدّ بعيد. سنتدرّج في فصوله بشكل سلس، حيث سنتناول موضوعات بسيطة في البداية، ثم ومع تقدّمنا في الفصول سنغطي موضوعات أوسع وأشمل.

ستلاحظ وجود تطبيقات بسيطة في كل فصل، نشرح من خلالها فكرة أو أفكار محدّدة. ولكن في نهاية الكتاب سنختم بتطبيقين عمليين، يوضّحان معظم الأفكار التي تفتت تغطيتها مسبقًا. الهدف من الكتاب هو أن تنطلق مع Vue.js بسهولة ويُسر، وبالتالي فهو كتاب مرجعيّ شامل لإطار العمل Vue.js. سافترض أنه لديك معرفة جيدة بلغة JavaScript و بلغة HTML أيضًا، كما ويُفضّل أن تمتلك معرفة أساسية بلغة CSS.

سنطرق في هذا الفصل إلى المواضيع التالية:

- ماهو إطار العمل framework في JavaScript؟
- تطبيقك الأول مع Vue.js
- إضافة مزايا جديدة لتطبيقنا الأول
- العمل على حاسوب محليّ
- تمارين داعمة

## 1.1 ماهو إطار العمل framework في JavaScript؟

ببساطة و بمعزل عن أي تفاصيل تقنية، يساعدنا إطار العمل عمومًا في إنشاء تطبيقات عصرية حديثة، حيث يحتوي على الأدوات اللازمة لتسهيل حياة المبرمجين في إنشاء تطبيقات معقدة بزمان قياسي، فلا يحتاج المبرمج أن يستخدم الأسلوب التقليدي القديم في التعامل مع نموذج كائنات المستند DOM باستخدام JavaScript، حيث يمكن باستخدام إطار العمل التعامل مع العناصر في DOM بشكل قياسي ومجرد وسهل إلى حد بعيد.

بدأ العمل مع JavaScript بتطوير مكتبات شهيرة مثل **jQuery** و **Mootools**، حيث سهّلت مثل هذه المكتبات العمل مع JavaScript على نحو كبير، وقلّلت مشاكل التوافقية بين المتصفحات المختلفة التي لم تكن تتبع معيارًا موحدًا.

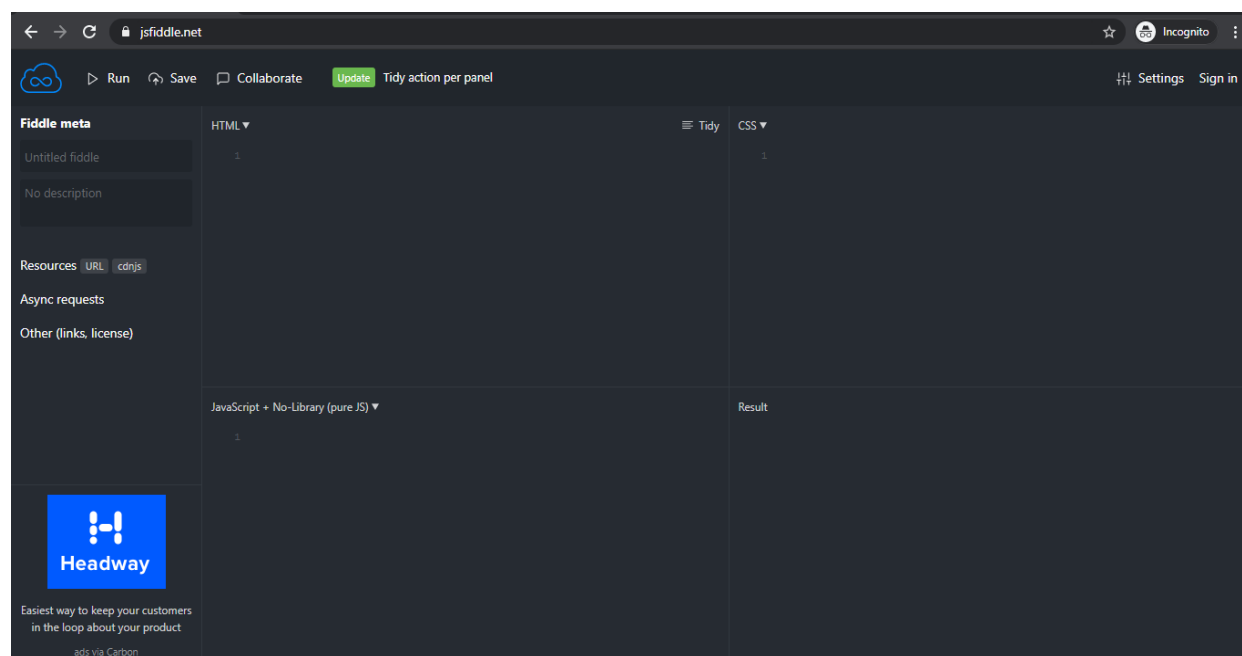
تطوّرت الأمور بعد ذلك لتبدأ أطر عمل متكاملة في الظهور يمكن أن نقسمها إلى مرحلتين أساسيتين. المرحلة الأولى كانت مع ظهور أطر عمل مثل **Backbone** و **Ember** و **Knockout** و **AngularJS**. أما المرحلة الثانية وهي المرحلة الحالية فكانت مع أطر عمل مثل **React** و **Angular** و **Vue**.

تعدّ **Vue.js** من أحدث أطر العمل في JavaScript، وهي إطار عمل واعد، يجمع بين السهولة والمرونة والقوة، بالإضافة إلى صغر حجم الملف الخاص بها (فقط 26 كيلو بايت) مما يجعل تحميل صفحات الويب سهلة إلى حد بعيد. سنهتم بالإصدار 2 من **Vue.js** مع التأكيد بأنّ ما سنأخذه هنا يمكن تطبيقه بسهولة في إصدارات لاحقة، إذ نزل الإصدار 3 بعد الانتهاء من الكتاب ولكن ذُكر أنه عملية إعادة كتابة للإطار وليس فيه اختلافات جوهرية عن الإصدار 2 السابق.

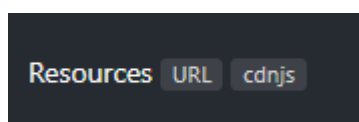
## 1.2 تطبيقك الأول مع Vue.js

قبل البدء باستخدام **Vue.js** علينا أن نحصل على الملف الخاص به. لدينا عدّة خيارات لذلك، سأختار أحد أسهل الخيارات وهو الحصول على الملف الخاص بإطار العمل عن طريق شبكة تسليم محتوى **CDN**، ثم نتحدّث عن الخيارات الأخرى فيما بعد. يمكنك استخدام **رابط CDN** التالي للحصول على **Vue.js**؛ الرابط هو [unpkg.com/vue@2.6.11/dist/vue.js](https://unpkg.com/vue@2.6.11/dist/vue.js) سنحتاج حاليًا إلى الرابط فقط.

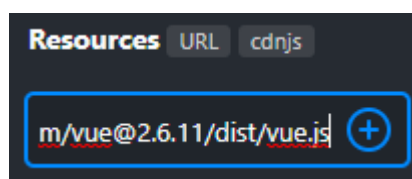
انتقل الآن إلى الموقع [jsfiddle.net](https://jsfiddle.net) الذي يوفر بيئة تطوير بسيطة وممتازة لتجريب **Vue.js** دون تحميل أي شيء على حاسوبك. ستحصل على شكل شبيه بما يلي:



انقر على الرابط URL الذي يظهر في القسم الأيسر في الأعلى من النافذة السابقة، بجوار كلمة Resources:



بعد نقر الرابط السابق ستحصل على مربع نص، انسخ فيه رابط CDN السابق، ثم انقر الزر المجاور الذي يظهر على شكل إشارة زائد:



نكون بذلك قد أخبرنا بيئة التطوير المصغرة أننا بصدد استخدام الملف الخاص بإطار العمل Vue.js.

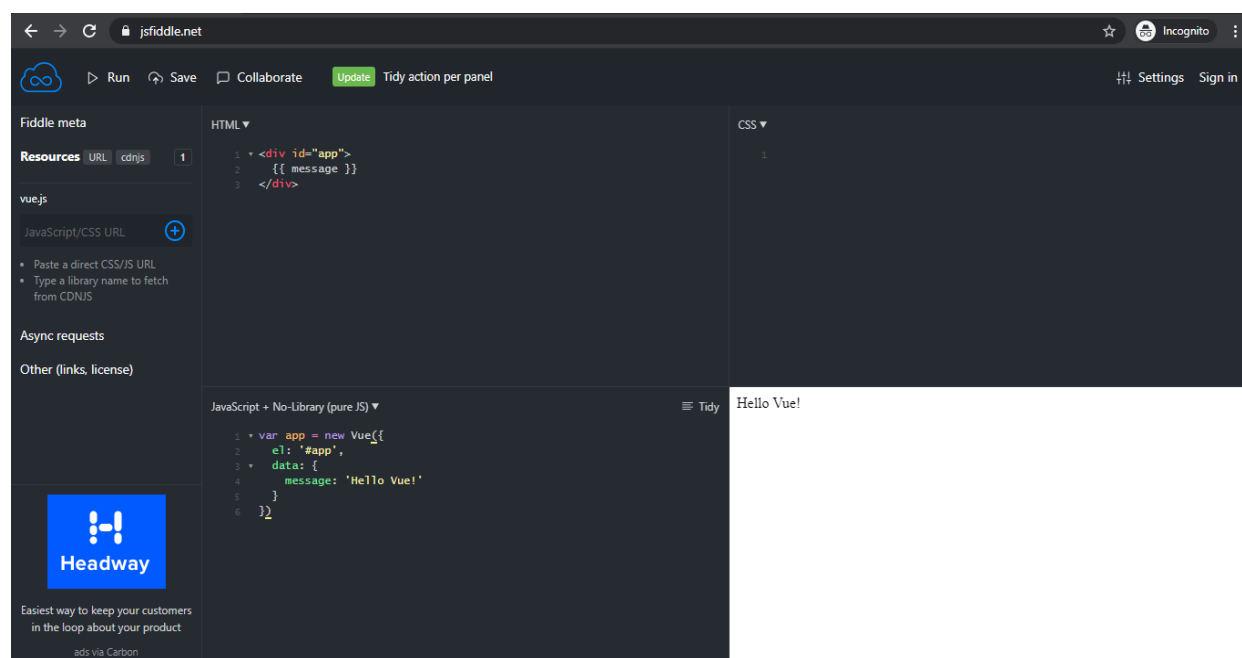
انسخ الآن شيفرة HTML التالي إلى القسم الخاص بأكواد HTML من الصفحة الرئيسية لموقع jsfiddle:

```
<div id="app">
  {{ message }}
</div>
```

ثم انسخ شيفرة JavaScript التالية والصقها ضمن القسم الخاص بشيفرات JavaScript في الصفحة الرئيسية الموقع:

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

انقر الآن زر التشغيل Run الموجود في الزاوية اليسرى العليا من الصفحة. إذا نفذت الخطوات السابقة بدقة، ستلاحظ ظهور العبارة Hello Vue! في القسم الأيمن الأسفل من الصفحة. مبارك! لقد بنيت تطبيق Vue.js الأول لك.



دعنا الآن نحلل ما قمنا به حتى الآن. أتوقع أنّ شيفرة HTML التي كتبناها سهلة وواضحة، حيث أنشأنا عنصر <div> وأسندنا له المعرّف app، ووضعنا ضمن وسمي الفتح والإغلاق له التعبير التالي:

```
{{ message }}
```

هذا التعبير هو الأساس في Vue.js، سنتحدث عنه بعد قليل. الآن لاحظ معي شيفرة JavaScript التالية:

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

أعتقد أنّ بداية هذه الشيفرة مألوفة، حيث أننا نعمل على إنشاء كائن من النوع Vue باستخدام الكلمة المفتاحية new. لاحظ معي ماذا نمزّر إلى النوع Vue:

```
{
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
}
```

المقطع السابق عبارة عن كائن آخر يُسمى بكائن الخيارات (options instance) يحوي الإعدادات التي نريد أن تكون ضمن كائن Vue.js الجديد. ففي الحقل `el` (يمثل أول حرفين من كلمة `element` أي العنصر) تُسند القيمة `'#app'` والتي تمثل معرف عنصر HTML المُستهدف (في حالتنا هذه هو عنصر `div` الذي أنشأناه قبل قليل)، أما الحقل `data` فيتم إسناد كائن آخر إليه يحتوي على حقل واحد وهو `message` وقيمته `'Hello Vue!'`. لاحظ أن الكلمة `'message'` هي نفسها الموجودة ضمن زُماز HTML السابق الموضوعه بحاضنة مزدوجة `{{ message }}`. وهكذا فعند تنفيذ البرنامج السابق سيعمل Vue على استبدال التعبير `{{ message }}` ضمن عنصر HTML الذي يحمل المعرف `app` بالنص `'Hello Vue!'` وهذا كل شيء.

رغم وجود من يميز بين الكائن `object` والنسخة `instance`، إلا أنني لا أفترق بينهما من الناحية العملية. لذلك تراني أستخدم "كائن" دومًا.

حاول الآن إجراء بعض التغييرات على النص السابق، وأعد التنفيذ باستخدام الزر Run لتشهد التغييرات الجديدة على الخرج.

في الحقيقة إنَّ ما جرى في هذا المثال يتعدى عن كونه مجرد عملية استبدال تعبير برسالة مجهزة مسبقًا، إن ما يجري من وراء الكواليس هو عملية ربط كامل بين الحقل `message` الموجود ضمن شيفرة JavaScript وبين التعبير `{{ message }}` الموجود ضمن زُماز HTML وسنرى كيف ذلك في الفقرة التالية.

## 1.3 إضافة مزايا جديدة لتطبيقنا الأول

حان الوقت لإجراء بعض التحسينات على تطبيقنا الأول. انتقل إلى قسم HTML وأضف عنصر إدخال نصي `input` على النحو التالي:

```
<div id="app">
  <input type='text' v-on:input="updateInfo"/>
  {{ message }}
</div>
```

لقد وضعت عنصر الإدخال فوق التعبير `{{ message }}` مباشرة. الشيء الجديد هنا هو السمة (أي attribute) التي تدعى `v-on:input` وهي بحد ذاتها مكونة من قسمين الموجّه directive وهو: `v-on` والوسيط `input` المراد تمريره إلى الموجّه. يلعب الموجّه `v-on` دور مُنصّت حدث `event listener`. وفي حالتنا هذه فإننا نرغب بالإنصات لحدث الإدخال `input` (معامل parameter) لعنصر الإدخال النصي. لاحظ القيمة المُسندة الموجّه: `updateInfo` وهي بكل بساطة اسم التابع الذي سيتم استدعاؤه في حال تمّ إجراء أي إدخال ضمن عنصر الإدخال النصي من قِبَل المستخدم. سنعرّف هذا التابع الآن.

استبدل الشيفرة القديمة الموجودة بالقسم المخصّص لشيفرات JavaScript في الصفحة الرئيسية للموقع بالشيفرة التالية:

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  },
  methods: {
    updateInfo: function(event) {
      this.message = event.target.value;
    }
  }
})
```

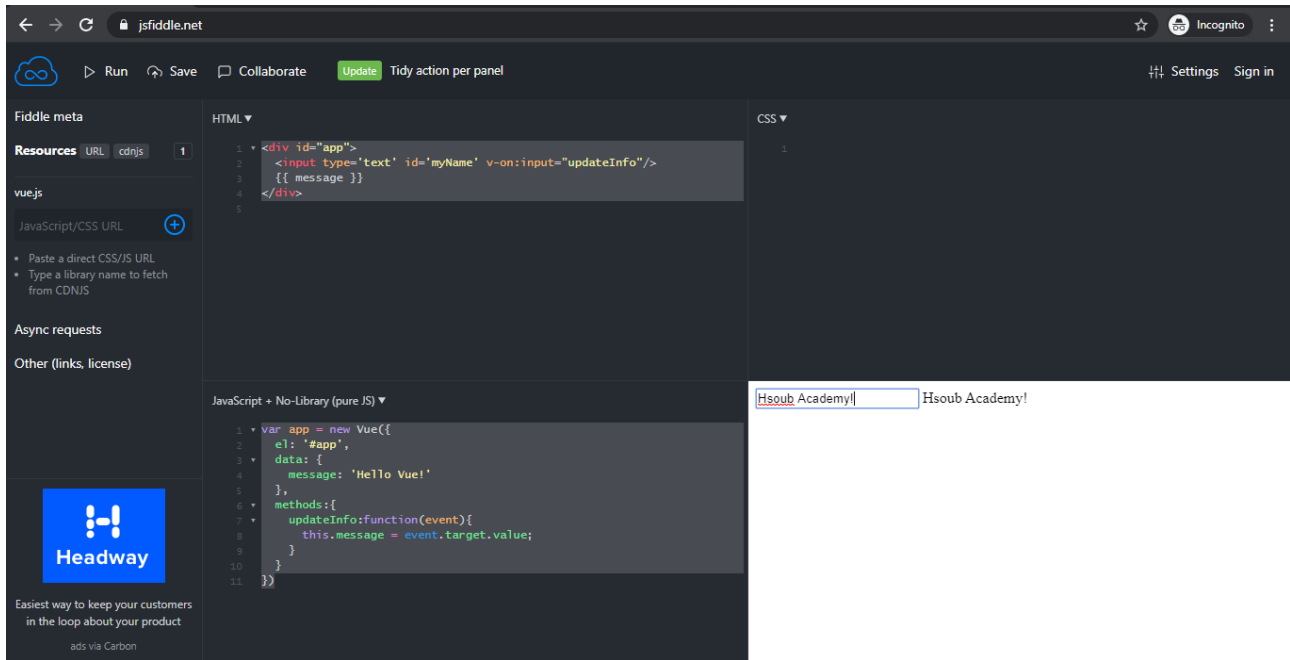
الفرق الوحيد بين المقطعين الجديد والقديم، أننا أضفنا القسم `methods` الذي يتم فيه تعريف جميع التوابع (`methods`) التي نرغب باستخدامها. واضح أننا سنعرّف التابع `updateInfo` الذي تحدثنا عنه قبل قليل. عرّفنا هذا التابع على أنّه دالة `function` ومزّرنّا إليه الكائن `event` وهو كائن يحتوي على بيانات الحدث الذي تمّ توليده تلقائيًا بسبب كتابة المستخدم ضمن مربّع النص. العبارة البرمجية الوحيدة الموجودة في هذا التابع هي:

```
this.message = event.target.value;
```

على نحو غريب قليلاً، ترمز الكلمة `this` هنا إلى الحقل `data` (المعرّف ضمن نفس كائن Vue.js) والذي يحتوي بدوره على الحقل `message`. سنوضّح سبب ذلك في فصل لاحق. القسم الثاني من العبارة بسيط، وفيه نستخلص البيانات التي أدخلها المستخدم من خلال التعبير `event.target.value`.

جرب تنفيذ البرنامج الآن بنقر زر Run، ثم حاول كتابة أي شيء ضمن مربع النص، ستلاحظ أنّه سيتم تحديث الرسالة بشكل فوري بالنص الذي تكتبه!





## 1.4 العمل على حاسوب محلي

إذا لم ترغب أن تعمل على jsfiddle.net يمكنك العمل بالتأكيد في وضع عدم اتصال محلياً على حاسوبك الشخصي. علينا أولاً تنزيل Vue.js عن طريق الرابط [vuejs.org/js/vue.js](https://vuejs.org/js/vue.js) (من الممكن أن يظهر محتوى الملف كاملاً ضمن صفحة عادية في المتصفح، انسخ محتويات الصفحة واحفظها ضمن جديد، سقّه: vue.js). في الحقيقة يمكنك استخدام هذه النسخة لأغراض التطوير البرمجية فقط، أما عندما يصبح التطبيق قيد الاستخدام العملي فيُنصح باستخدام النسخة التالية: [vuejs.org/js/vue.min.js](https://vuejs.org/js/vue.min.js)

حصلت على النسختين السابقتين من الموقع الرسمي لإطار العمل Vue.js من الصفحة التالية: [vuejs.org/v2/guide/installation.html](https://vuejs.org/v2/guide/installation.html)

أنشئ مجلداً وسقّه vuejs-first-app، ثم انسخ ضمن هذا المجلد ملف vue.js الذي نزلته قبل قليل. باستخدام محرر HTML المناسب لك، أنشئ ملف جديد واختر له مثلاً الاسم index.html. ثم احفظه في نفس المجلد السابق بجوار الملف vue.js، وبعد ذلك أضف إليه المحتوى التالي الذي قمت بتجميعه من محتويات التطبيق المحسن الذي تناولناه في الفقرة السابقة:

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Vue.js</title>
    <script src="vue.js"></script>
  </head>
```

```
<body>
  <div id="app">
    <input type='text' v-on:input="updateInfo"/>
    {{ message }}
  </div>

  <script type="text/javascript">
    var app = new Vue({
      el: '#app',
      data: {
        message: 'Hello Vue!'
      },
      methods: {
        updateInfo: function(event) {
          this.message = event.target.value;
        }
      }
    })
  </script>
</body>
</html>
```

لاحظ معي أننا قد أضفنا مرجعًا لملف `vue.js` المحلي عن طريق وسم `script` ضمن القسم `head` من المستند، وأضفنا شيفرة `JavaScript` المسؤولة عن تحديث خرج التطبيق ضمن القسم `body` من المستند. يمكنك متابعة كتابة تطبيقاتك على هذا النحو، ولو أنني أفضل استخدام منصات مثل [jsfiddle.net](https://jsfiddle.net) و [codepen.io](https://codepen.io) على الأقل في هذه المرحلة التي نسبر بها أغوار `Vue.js`.

## 1.5 ختام الفصل

تناولنا في هذا الفصل مقدمة سريعة إلى إطار العمل الواعد `Vue.js` حيث تعلّمنا كيف نبدأ مع `Vue.js` وكيف نكتب تطبيقات بسيطة توضّح لنا مدى السهولة والمرونة التي يتمتع بها إطار العمل هذا. سنتابع عملنا في الفصول اللاحقة لنسبر أغوار `Vue.js` بشكل سهل ومرتّج.

## 1.6 تمارين داعمة

سأزودك في التمرينين التاليين بمسألتين بسيطتين للتدرب على المفاهيم الموجودة في هذا الفصل. يوجد بعد كل تمرين حل مقترح، حاول ألا تنظر إلى الحلول المقترحة قبل تقديم حلولك.

## 1.6.1 تمرين 1

استخدم الوسيط `click` بدلاً من الوسيط `input` مع الموجه `v-on` لكي يستجيب التطبيق عندما ينقر المستخدم على عنصر الإدخال بدلاً من الكتابة داخله وذلك في التطبيق المحسن الذي تحدثنا عنه قبل قليل. اجعل التطبيق يعرض الرسالة "Clicked!" عندما ينقر المستخدم.

### الحل المقترح

الحل في هذا التمرين بسيط. استبدل الوسيط `input` بالوسيط `clicked` ضمن زُماز HTML ثم وفي القسم المخصص لشفرة JavaScript، استبدل بالسطر التالي:

```
this.message = event.target.value;
```

السطر:

```
this.message = 'Clicked!';
```

## 1.6.2 تمرين 2

في هذا التمرين سننشئ تطبيق آلة حاسبة بسيط يقوم بعملية حسابية واحدة وهي جمع عددين. أقترح الواجهة البسيطة التالية:

+  = 100

الميزة في هذا التطبيق أنه إذا بدأ المستخدم بالكتابة في أي مربع، يجب أن يقوم التطبيق بإيجاد المجموع بشكل فوري أثناء إدخاله الأرقام. لن نقوم في هذا التمرين بالتحقق من صحة إدخال المستخدم.

### الحل المقترح

طريقة التنفيذ مشابهة جدًا للتطبيقات التي تناولناها في هذا الفصل. سأضيف حقلين إضافيين فقط لتخزين قيمتي المعاملين الأيمن والأيسر لعملية الجمع. شيفرة HTML المقترحة هي:

```
<div id="app">
  <input type='text' v-on:input="updateLeftOper"/>
  <label>+</label>
  <input type='text' v-on:input="updateRightOper"/>
  <label>=</label>
  <label>{{result}}</label>
</div>
```

أما شيفرة JavaScript المقترحة فهي على الشكل التالي:

```
var app = new Vue({
  el: '#app',
  data: {
    result: 0,
    leftOper: 0,
    rightOper: 0
  },
  methods: {
    updateLeftOper: function(event){
      this.leftOper = parseFloat(event.target.value)
      this.result = this.leftOper + this.rightOper;
    },
    updateRightOper: function(event){
      this.rightOper = parseFloat(event.target.value)
      this.result = this.leftOper + this.rightOper;
    }
  }
})
```

إذا لم تستطع استيعاب الشيفرة السابقة تمامًا فلا بأس، سنتكلم المزيد عن هذه الأمور لاحقًا.

## 2. التعامل مع DOM

سنتعلم في هذا الفصل:

- فهم قوالب Vue.js
- الوصول إلى البيانات والتوابع من كائنات Vue.js
- الربط مع السمات Attributes
- كتابة شيفرة HTML خام
- التعامل مع الأحداث Events
- استخدام الربط ثنائي الاتجاه

نتابع عملنا في هذا الفصل وهو الفصل الثاني والذي سنتعلم فيه هذه المرة كيفية الوصول والتعامل مع DOM، حيث سنتعلم كيف نستخدم موجهات Vue.js مختلفة للوصول إلى بيانات كائن Vue.js والتفاعل معها، وسنتوسع في التعامل مع الأحداث events بالإضافة إلى كيفية استخدام الربط ثنائي الاتجاه مع العناصر.

## 2.1 فهم قوالب Vue.js

تعاملنا في الفصل السابق مع تطبيقات استخدمت مزايا بسيطة من Vue.js، وإذا كنت تذكر أننا قد كتبنا شيفرة HTML بسيطة ومن ثم استخدمنا الاستبدال النصي `{{ message }}` واستخدمنا أيضًا موجه `v-on` للاستجابة إلى دخل المستخدم.

ما يقوم به Vue.js من وراء الكواليس، هو أخذ نسخة عن شيفرة HTML وحفظها داخليًا على شكل قالب `template`، بعد ذلك يتم إجراء عملية تصيير (`rendering`) على نسخة من القالب السابق، باستخدام الموجهات وتعابير الاستبدال النصي (في حال وجودها ضمن القالب)، ثم بعد الانتهاء من التصيير يتم عرض الخرج النهائي على المستخدم.

الآن، وعندما يُحدث أي تغيير جديد في قيمة أي حقل مرتبط من كائن Vue.js، ستُجرى عملية تصيير جديدة على نسخة جديدة من القالب السابق المخزن داخليًا، ثم يُعرض الخرج النهائي من جديد على المستخدم.

أي كما لو أننا أنشأنا ارتباطًا دائمًا بين كائن Vue.js وبين شيفرة HTML. وهذا ما رأيناه فعليًا في التطبيقات البسيطة التي تناولناها في الفصل السابق.

من باب التذكير، نستخدم في هذا الكتاب الموقع [jsfiddle.net](https://jsfiddle.net) بشكل افتراضي لتشغيل جميع التطبيقات التي نكتبها. ونحتاج بالطبع إلى إدراج ملف إطار العمل Vue.js لكي نستطيع تنفيذ هذه التطبيقات. إذا أردت أن تعرف كيف ذلك، يمكنك العودة إلى الفصل السابق.

## 2.2 الوصول إلى البيانات والتوابع من كائنات Vue.js

انظر إلى المثال التالي (مثل العادة، أول مقطع يمثل شيفرة HTML وثاني مقطع يمثل شيفرة JavaScript):

```
<div id="app">
  {{ title }}
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    title: 'Hello Vue!'
  }
})
```

```
}
})
```

عندما نستخدم الاستبدال النصي `{{ title }}` كما وسبق أن فعلنا مسبقًا، لا نستخدم الكلمة `this` قبل `title` كما هو واضح. في الحقيقة أن أي كلمة تُشير إلى حقل (مثل `title`) موجودة ضمن حاضنة مزدوجة، سيتم اعتبارها على أنها حقل ضمن القسم `data` في كائن `Vue.js` الموافق. الآن عند تنفيذ التطبيق السابق سيؤدي إلى ظهور الجملة `Hello Vue!` كما هو متوقع.

وبالمثل أيضًا، يمكننا في الواقع استخدام تابع مثل `displayMessage()` ليحقق نفس الخرج السابق تمامًا، وبنفس الأسلوب تقريبًا. استخدم المثال التالي:

```
<div id="app">
  {{ displayMessage() }}
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    title: 'Hello Vue!'
  },
  methods: {
    displayMessage: function(){
      return this.title;
    }
  }
})
```

لاحظ أننا استخدمنا هذه المرة التابع `displayMessage()` ضمن الحاضنة المزدوجة `{{ displayMessage() }}` ومرة أخرى لم نستخدم الكلمة `this` قبل اسم التابع. أي تابع يُكتب بهذه الطريقة سيعتبر افتراضيًا على أنه تابع موجود ضمن القسم `methods` من كائن `Vue.js`. ولكن هذا السلوك الافتراضي لا يسري على شيفرة `JavaScript` الموجودة ضمن كائن `Vue.js` حيث يجب استخدام الكلمة `this` في كل مرة أردنا فيها الوصول إلى أحد أعضاء كائن `Vue.js`.

## 2.3 الربط مع السمات Attributes

لا يمكن استخدام تقنية الاستبدال النصي (باستخدام الحاضنة المزدوجة) لإدراج قيم ضمن سمات العناصر. لفهم هذا الموضوع بشكل جيّد، انظر معي إلى المثال التالي:

```
<div id="app">
```

```
<p> {{ message }} - <a href='{{link}}'>Hsoub Academy</a> </p>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!',
    link: 'https://academy.hsoub.com/'
  }
})
```

عند تنفيذ التطبيق السابق في jsfiddle.net ستحصل على الخرج التالي:

Hello Vue! - [Hsoub Academy](https://academy.hsoub.com/)

لاحظ أنه خرج منسق كما هو متوقع، ولكن إذا جربت النقر على الرابط لن ينقلك إلى موقع أكاديمية حسوب كما هو متوقع، في الحقيقة سينقلك هذا الرابط إلى صفحة ضمن نفس موقع jsfiddle.net وهذه الصفحة بالطبع ستكون غير موجودة.

السبب في ذلك أن تقنية الاستبدال النصي تُعامل محتويات الحقل link كنص مجرد، يمكنك ملاحظة الرابط الناتج بعد النقر: [fiddle.jshell.net/\\_display/%7B%7Btitle%7D%7D](https://jsfiddle.net/_display/%7B%7Btitle%7D%7D) الحل لهذه المشكلة بسيط، ويتمثل في تجنّب استخدام السمة href بهذا الشكل، إنما ينبغي استخدام الموجه v-bind مع الوسيط href على النحو التالي:

```
v-bind:href = 'link'
```

حيث link هو نفسه الحقل الموجود ضمن كائن Vue.js. استبدل بالتعبير السابق السمة href القديمة الموجودة ضمن شيفرة HTML الموجود في المثال السابق، بعد الاستبدال سيصبح شكل شيفرة HTML على النحو التالي:

```
<div id="app">
  <p> {{ message }} - <a v-bind:href = 'link'>Hsoub Academy</a> </p>
</div>
```

أعد تشغيل التطبيق مرة أخرى، ستحصل على نفس الخرج، ولكن هذه المرة إذا نقرت على الرابط ستنتقل إلى موقع أكاديمية حسوب. إذاً كخلاصة على ما سبق، إذا أردت أن تربط مع السمات فعليك استخدام موجه الوسيط المناسب بدلاً من استخدام تقنية الاستبدال النصي. سنتناول عددًا من هذه الموجهات خلال هذا الكتاب، ويمكنك دومًا زيارة [الصفحة الرسمية](#) لإطار العمل Vue.js للاطلاع على جميع الموجهات المتوفرة.



## 2.4 كتابة شيفرة HTML خام

نحتاج في بعض الأحيان إلى كتابة شيفرة HTML خام مباشرة في الصفحة. قد يتبادر إلى ذهنك أن تستخدم الاستبدال النصي مع الحاضنة المزدوجة، ولكن لن تنفع هذه التقنية في هذه الحالة. تأمل معي المثال التالي لفهم أفضل حول هذه المشكلة:

```
<div id="app">
  {{ raw }}
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    raw: '<ul><li>First Item</li><li>Second Item</li><li>Third
Item</li></ul>'
  }
})
```

الهدف من التطبيق السابق هو عرض قائمة غير مرتبة عن طريق العنصر `ul` تظهر ثلاثة عناصر فقط: `First Item` و `Second Item` و `Third Item`. ولكن عند التنفيذ لن تحصل على ما هو متوقع، ستحصل على الخرج التالي:

```
<ul><li>First Item</li><li>Second Item</li><li>Third Item</li></ul>
```

أي أنك ستحصل على نص عادي دون أن يتعرّف عليه المتصفح على أنه شيفرة HTML. يمكن هذه المشكلة بسهولة بإجراء تعديل على شيفرة HTML فقط على النحو التالي:

```
<div id="app">
  <p v-html='raw'>

  </p>
</div>
```

التعديل الذي أجرته هو استخدام الموجه `v-html` الذي يسمح في حالتنا هذه باستخدام محتويات الحقل `raw` كـ شيفرة HTML نظامية وليس مجرد نص عادي (لاحظ أنني قد تخلصت من الاستبدال النصي `{{ raw }}`. أعد تنفيذ التطبيق، لتحصل على قائمة مُنسقة بشكل صحيح.

## 2.5 التعامل مع الأحداث (Events)

للأحداث كما نعلم أهمية عظيمة في تطوير التطبيقات التي تتفاعل مع المستخدم. تدعم Vue.js الأحداث بشكل جيد، ولقد تعاملنا في الفصل السابق مع نوعين من الأحداث: حدث الإدخال ضمن مربع النص، وحدث النقر بزر الفأرة على عنصر HTML. سنتناول في هذا الفصل الأحداث بشيء من التفصيل، حيث سنتعلم كيف ننصت إلى أحداث الفأرة، بالإضافة إلى الإنصات إلى أحداث لوحة المفاتيح.

### 2.5.1 الإنصات إلى أحداث الفأرة

نبدأ بالتعامل مع أحداث الفأرة، لثعش ذاكرتك، انظر معي إلى التطبيق البسيط الموجود في الفصل السابق:

```
<div id="app">
  <input type='text' v-on:input="updateInfo"/>
  {{ message }}
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  },
  methods: {
    updateInfo: function(event){
      this.message = event.target.value;
    }
  }
})
```

سبق وأن ذكرنا في الفصل السابق أنه يتم الإنصات إلى أي حدث ضمن عنصر ما باستخدام الموجه `v-on`، حيث يتم تغيير الوسيط المُمرّر لهذا الموجه بتغيير نوع الحدث المراد الإنصات له. بعد تحديد نوع الوسيط المراد تمريره للموجه `v-on` يتم تحديد التابع الذي سيستجيب (سيعالج) هذا الحدث، وهو عبارة عن تابع ضمن القسم `methods` ضمن كائن `Vue.js` يتم استدعاؤه عند وقوع الحدث.

بالنسبة للتابع المعالج للحدث (التابع `updateInfo` في المثال السابق) سيتم توليد كائن يحتوي على معلومات مهمة حول الحدث الذي وقع، ويتم تمرير هذا الكائن بشكل تلقائي إلى التابع المعالج للحدث.

على العموم، يمكن الاستغناء عن هذا السلوك التلقائي، وتمرير قيمة كيفية للتابع المعالج للحدث. انظر معي إلى المثال التالي:

```
<div id='app'>
  <button v-on:click="increase(2)">Increase!</button>
  <p>{{counter}}</p>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    counter: 0
  },
  methods: {
    increase: function(value){
      this.counter += value;
    }
  }
})
```

الشفيرة السابقة مألوفة، مع ملاحظتين جديدتين. الأولى أننا لم نكتفي بكتابة اسم التابع المعالج للحدث ضمن الموجه `v-on:click` فحسب، إنما قد مَرَرنا القيمة 2 كوسيط لهذا التابع على الشكل: `increase(2)` (لاحظ شيفرة HTML). بالمقابل، إذا تأملت شيفرة JavaScript ضمن تعريف التابع `increase` في القسم `methods`، فستلاحظ أننا نعامل الوسيط `value` كمتغير يحمل قيمة عددية وليس ككائن يحمل معلومات حول الحدث الذي وقع. أي أننا قد استطعنا تغيير السلوك الافتراضي لعملية استدعاء التابع المعالج. بالنسبة للمثال السابق، فكما هو واضح، يعمل التطبيق على زيادة قيمة المتغير `counter` بمقدار القيمة `value` (في مثالنا السابق ستكون تساوي 2) في كل مرة يتم فيها نقر الزر.

في بعض الحالات قد نحتاج إلى تمرير كائن الحدث بالإضافة إلى تمرير قيمة كيفية بنفس الوقت. تدعم Vue.js هذا الأمر ببساطة من خلال تمرير الكلمة المحجوزة `$event` إلى التابع المعالج بالإضافة إلى القيمة الكيفية المراد تمريرها. إذا أردنا تطبيق ذلك على المثال الأخير فسيصبح تعريف الزر `button` على النحو التالي:

```
<button v-on:click="increase(2, $event)">Increase!</button>
```

وبالنسبة لتعريف التابع المعالج ضمن القسم `methods` فسيصبح على النحو التالي:

```
increase: function(value, event){
    this.counter += value;
}
```

أي مجزء إضافة وسيط آخر.

## 2.5.2 التعديل على كيفية الاستجابة للأحداث

نحتاج في بعض الأحيان أن نُعدّل على كيفية الاستجابة للأحداث، فربما نحتاج في وقت ما إلى إيقاف الاستجابة لحدث ما لأحد العناصر دونًا عن العناصر الأخرى في HTML. لكي أضرب لك مثالاً جميلاً حول هذا الأمر، اسمح لي أولاً أن أقدم لك حدث حركة الفأرة `mousemove`. يُولّد هذا الحدث عند مرور مؤشر الفأرة فوق عنصر ما، ويتم استخدامه كما هو متوقّع مع الموجه `v-on`. انظر إلى المثال البسيط التالي:

```
<div id="app">
  <p v-on:mousemove='updateCoordinates'>
    Mouse cursor at: ({{x}}, {{y}})
  </p>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    x:0,
    y:0
  },
  methods:{
    updateCoordinates:function(event){
      this.x = event.clientX;
      this.y = event.clientY;
    }
  }
})
```

جَرَّب تنفيذ التطبيق البسيط السابق، ثم حرّك الفأرة فوق العنصر الوحيد الظاهر أمامك. ستحصل على خرج شبيه بما يلي:

Mouse cursor at: (123, 20)

الجديد هنا هو استخدام الموجه `v-on:mousemove` حيث أسندنا إليه التابع `updateCoordinates` المعرّف بطبيعة الحال ضمن القسم `methods` في كائن `Vue.js`. لاحظ

معي أيضًا كيف نحصل على الإحداثيات الحالية لمؤشر الفأرة (الفاصلة x والترتيب y) ضمن التابع `updateCoordinates`:

```
this.x = event.clientX;
this.y = event.clientY;
```

الآن إذا أردنا أن ننشئ منطقة "ميتة" (ضمن عنصر `span` مثلاً) ضمن عنصر `p` الذي يعرض الإحداثيات، بحيث لا يؤدي مرور مؤشر الفأرة فوق هذه المنطقة إلى توليد الحدث `mousemove`، فينبغي علينا عندها التعديل على الحدث `mousemove` كما يلي (سألون التعديلات الإضافية بالأخضر):

```
<div id="app">
  <p v-on:mousemove='updateCoordinates'>
    Mouse cursor at: ({{x}}, {{y}}) -
    <span v-on:mousemove='uncoveredArea'>Uncovered Area</span>
  </p>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    x:0,
    y:0
  },
  methods:{
    updateCoordinates:function(event){
      this.x = event.clientX;
      this.y = event.clientY;
    },
    uncoveredArea: function(event){
      event.stopPropagation();
    }
  }
})
```

أضفت الموجه `v-on:mousemove` إلى العنصر `span` وأسندت المعالج `uncoveredArea` له.

بالنسبة للتابع `uncoveredArea` فقد أجريت تعديل على الحدث من خلال استدعاء التابع `stopPropagation()` من الكائن `event`. المعنى الحرفي لهذا التابع هو "إيقاف الانتشار" أي أننا سنعيق الاستجابة لهذا الحدث عندما يمر مؤشر الفأرة فوق العنصر `span`.

جرب تنفيذ التطبيق السابق، ولاحظ التغيير الذي سيحدث عندما يمر مؤشر الفأرة فوق عنصر `span`. إذا أردت الإحساس بالفرق، يمكنك أن تحذف التعليمة `event.stopPropagation()` ثم أعد تنفيذ التطبيق.

مرة أخرى، لتري كيف أنَّ الإحداثيات ستتغير عندما يمر مؤشر الفأرة فوق عنصر `span` هذه المرة. يمكن استخدام صيغة أبسط للتعديل على الأحداث، فمن الممكن حذف التابع `uncoveredArea` بالكامل من قسم `methods`، والاكتفاء بالقسم الخاص بالموّجه على النحو التالي:

```
v-on:mousemove.stop=''
```

أي أننا قد استغينا عن الشيفرة اللازمة لإيقاف انتشار الحدث `mousemove`. نسمي `stop`. هنا بمعدّل الحدث (event modifiers) وهناك عدّة معدّلات أحداث مفيدة سنستعرض بعضها في هذا الكتاب.

### 2.5.3 الإنصات إلى أحداث لوحة المفاتيح

يمكننا أحياناً أن نحتاج إلى الإنصات أيضاً إلى الأحداث الناشئة من لوحة المفاتيح. والأسلوب المتبع هنا، يشبه إلى حدّ كبير ما كنّا نفعله مع أحداث الفأرة. إذا أردنا مثلاً الإنصات إلى حدث تحرير مفتاح من لوحة المفاتيح يمكن أن نستخدم الوسيط `keyup` للموجه `v-on` على النحو التالي:

```
v-on:keyup='methodName'
```

حيث `methodName` هو اسم التابع المعالج للحدث `keyup` والذي يجب أن يُوضّع ضمن القسم `methods`. دعنا الآن نوّظف ذلك في مثال بسيط:

```
<div id="app">
  <input type='text' v-on:keyup='keyIsUp' />
  <p>
    {{message}}
  </p>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    message: ''
  },
  methods: {
    keyIsUp: function(event) {
      this.message = event.target.value;
    }
  }
})
```

يعمل هذا التطبيق البسيط على تحديث الحقل `message` كلّما تمّ تحرير مفتاح من لوحة المفاتيح، وبالتالي سيؤدّي ذلك إلى تحديث محتويات عنصر `p` ضمن الواجهة.

ولكن دعنا نتساءل، ماذا لو أردنا أن يستجيب المعالج `keyIsUp` كلما خُزِر مفتاح المسافة (`space`) فقط، وليس عند أي مفتاح يُحرّره المستخدم. الجواب ببساطة، هو في استخدام معدّل الحدث `space`. بعد `keyup`. أضف فقط الكلمة `space`. إلى `keyup` إلى المثال السابق. أي على النحو التالي:

```
v-on:keyup.space = 'keyIsUp'
```

أعد تنفيذ التطبيق لترى أنّ محتويات عنصر `p` أصبحت لا تُحدّث إلا بعد تحرير المفتاح `space`. يوجد بالطبع العديد من المعدّلات التي تمثّل جميع المفاتيح على لوحة المفاتيح، فهناك مثلاً `enter` و `tab` و `up` لمفتاح السهم العلوي، و `down` لمفتاح السهم السفلي وهكذا.

لمعدّلات أحداث لوحة المفاتيح الكثير من الفوائد، يتمثّل أبسطها في إرسال المحتوى الذي أدخله المستخدم بمجرد ضغطه للمفتاح `Enter`، أو إرسال البيانات مباشرةً بينما يكتبها المستخدم للحصول على مقترحات أثناء عملية الكتابة (كما يفعل محرّك البحث غوغل أثناء كتابة المستخدم للمفردات المراد البحث عنها). وغيرها الكثير من الاستخدامات.

## 2.6 استخدام الربط ثنائي الاتجاه

في معظم الأمثلة السابقة عمدنا إلى استخدام ربط باتجاه واحد، من الشيفرة إلى عنصر `HTML`. وفي بعض الحالات استطعنا أن نعكس هذا الأمر. أي استطعنا تعديل قيمة الحقل عن طريق الانصات إلى حدث الإدخال `v-on:input` وبالتالي معالج حدث مخصّص لهذه الغاية. ولكن توجد طريقة مباشرة وسهلة لإيجاد ربط ثنائي الاتجاه فعلي في `Vue.js` وذلك باستخدام الموجه `v-model` وبدون الحاجة إلى معالج حدث، كمال في المثال التالي:

```
<div id="app">
  <input type='text' v-model='name' />
  {{ name }}
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    name: 'Hello Vue!'
  }
})
```

التطبيق السابق بسيط، وهو يعمل على إجراء ربط ثنائي الاتجاه بين الحقل `name` وبين عنصر مربع النص، أي سيكون هناك ارتباط آني بين الحقل `name` وبين عنصر مربع النص، فإذا حدث تغيير لأحدهما سينعكس مباشرةً على الآخر. نَقُذ التطبيق السابق وسترى مباشرةً الخرج التالي:

Hello Vue! Hello Vue!

لاحظ كيف أنَّ محتوى مربع النص قد تمَّت تعبئته تلقائيًا بقيمة الحقل `name`، وبالمثل إذا حاولت الآن كتابة أي شيء ضمن مربع النص سيتم تعديل قيمة الحقل `name` فورًا وفقًا له، وبالتالي سيُعدَّل محتوى النص الموجود في الطرف الأيمن بسبب وجود الاستبدال النصي `{{name}}`.

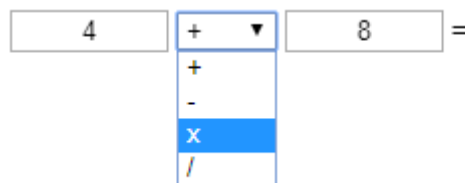
## 2.7 ختام الفصل

تعلَّمنا في هذا الفصل كيفية التعامل مع DOM، حيث تحدثنا عن القوالب، وكيف نصل إلى البيانات والتوابع بشيء من التفصيل، كما تعلَّمنا كيفية الربط مع السمات، والتعامل مع الأحداث، وتعلَّمنا كيفية الربط ثنائي الاتجاه الذي يسمح لنا بمزامنة البيانات بالاتجاهين.

## 2.8 تمارين داعمة

### 2.8.1 تمرين 1

يُطلب في هذا التمرين تطوير تطبيق الآلة الحاسبة البسيط الذي بنيناه في الفصل السابق. بحيث يسمح التطبيق الجديد بإجراء العمليات الحسابية الأربع بدلاً من عملية الجمع الوحيدة التي كان يدعمها التطبيق السابق. أقترح الواجهة التالية للتطبيق:



لاحظ أنَّني قد استخدمت عنصر `select`، يمكنك استخدام أي طريقة أخرى لاختيار العمليات الحسابية الأربع.

من الضروري أن يُوجد التطبيق الناتج النهائي إذا حدث أحد الأمرين التاليين:

- تغيير قيمة أحد المعاملين على طرفي العملية الحسابية.



- تغيير العملية الحسابية عن طريق القائمة المنسدلة.

أرجو أن يتمكن التطبيق من تمييز حالة القسمة على صفر، وإظهار رسالة مناسبة للمستخدم.

## 2.8.2 تمرين 2

يُطلب في هذا التمرين إنشاء تطبيق يقدم للمستخدم مقترحات نصية بينما يكتب المستخدم ضمن مربع نص. سنحاكي عملية الاتصال مع خادم بعيد عن طريق استخدام مصفوفة نصية ضمن الشيفرة. وسأعتبر أن المستخدم يحاول أن يدخل اسم دولة عربية، فتظهر قائمة المقترحات بالأسفل بينما تتم عملية الإدخال. كما في الشكل التالي:

الس

السعودية
السودان

ستحتاج بالطبع إلى التعامل مع أحداث لوحة المفاتيح، ولكي يكون الأمر أكثر سهولة بالنسبة إليك، يمكنك استخدام المصفوفة الجاهزة التالية كمصدر للبيانات التي يُفترض أن تكون قادمة من الخادوم:

```
[
  'السعودية',
  'البحرين',
  'مصر',
  'السودان',
  'ليبيا',
  'الجزائر',
  'المغرب',
  'تونس',
  'موريتانيا',
  'العراق',
  'سوريا',
  'لبنان',
  'قطر',
  'الإمارات',
  'الصومال',
  'جزر القمر',
  'الكويت',
  'سلطنة عُمان',
  'الأردن',
  'اليمن',
  'فلسطين'
]
```

# 3. الموجهات الشرطية والتكرارية

سنتعلم في هذا الفصل:

- كتابة شفرات JavaScript مباشرة ضمن القوالب
- التصيير الشرطي باستخدام `v-if` و `v-else` و `v-else-if`
- الفرق بين `v-if` و `v-show`
- تصيير القوائم باستخدام `v-for`
- المرور على خاصيات كائن

نتابع عملنا في هذا الفصل وهو الفصل الثالث، سنتعلم فيه كيفية كتابة شفرات JavaScript مباشرة ضمن القوالب دون الحاجة إلى الاستعانة بالتوابع كما كنا نفعل من قبل، بالإضافة إلى استخدام التصيير (rendering) الشرطي باستخدام `v-if` وأخواته `v-else` و `v-else-if`. كما سنتحدث عن الفرق بين `v-if` و `v-show` اللذان لهما نفس التأثير من الناحية الشكلية، ونختتم الفصل بالحديث عن التكرار باستخدام `v-for`.

هيا بنا لنسبر أغوار Vue.js!

### 3.1 كتابة شفرات JavaScript مباشرة ضمن القوالب

يمكننا في الكثير من الأحيان أن نكتب شيفرة JavaScript مباشرةً ضمن القوالب، دون الحاجة إلى إنشاء تابع خاص ووضعه في قسم `methods`. نلجأ إلى هذا الأسلوب، في حال كانت الشيفرة قصيرة ولا تحتوي على الكثير من التعقيد، بالإضافة إلى ضرورة أن يكون الناتج النهائي للشيفرة عبارة عن تعبير (`expression`). انظر معي إلى المثال التالي:

```
<div id="app">
  <span>Current temperature:</span><input type="text" v-
model='temperature' />
  <p>
    {{temperature > 35 ? 'Hot': temperature < 20 ? 'Cold' : 'Moderate'
  }}
  </p>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    temperature: 30
  }
})
```

يعمل التطبيق السابق على عرض مربع إدخال للمستخدم، حيث يعرض عليه إدخال درجة الحرارة الحالية، ثم يُقيم دخل المستخدم آنياً أثناء الكتابة على النحو التالي:

- إذا كانت درجة الحرارة أكبر من 35، سيعرض التطبيق الرسالة Hot
  - إذا كانت درجة الحرارة أصغر من 20، سيعرض التطبيق الرسالة Cold
  - إذا لم يتحقق الشرطان السابقان سيعرض التطبيق الرسالة Moderate أي معتدل.
- إذا نظرت إلى قسم HTML فستجد الشيفرة التالية ضمن الاستبدال النصي:

```
temperature > 35 ? 'Hot': temperature < 20 ? 'Cold' : 'Moderate'
```

شيفرة JavaScript هذه عبارة عن تعبير بسيط يستخدم العامل (operator) الثلاثي `??` بشكل متداخل لاختبار الحالات الثلاث السابقة.

لاحظ معي أننا قد استخدمنا هنا الربط ثنائي الاتجاه 'v-model' لربط قيمة الحقل temperature مباشرة بدخل المستخدم (يمكنك العودة للفصل السابق لمراجعة هذا الموضوع). لاحظ أيضًا أن كائن Vue.js في قسم JavaScript بسيط للغاية ولا يحتوي على أية توابيع.

## 3.2 التصيير الشرطي باستخدام v-if و v-else و v-else-if

نحتاج في الكثير من الأحيان إلى إخفاء جزء من الصفحة في حال تحقق (أو عدم تحقق) شرط ما. يمكن تحقيق هذا الأمر بسهولة في Vue.js من خلال استخدام الموجه v-if وأخواته. لنستخدم أولاً الموجه v-if لوحده:

```
<div id="app">
  <button v-on:click="show = !show">
    Click this!
  </button>
  <p v-if="show">
    Welcome to Hsoub Academy!
  </p>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    show: true
  }
})
```

التطبيق السابق بسيط، تتكوّن الواجهة من زر عادي ورسالة. عند نقر هذا الزر بشكل متكرر، تظهر الرسالة في الأسفل أو تختفي وفقًا لذلك.

نلاحظ أولاً الحقل show المعرّف ضمن قسم data، والذي أسندت إليه القيمة 'true' افتراضيًا. بالنسبة لشفيرة HTML لاحظ الموجه v-on:click وكيف أسندت إليه شيفرة JavaScript مباشرة:

```
v-on:click = "show = !show"
```

اعتدنا مسبقًا على إدراج تابع معالج للحدث، ولكن من الممكن كتابة شيفرة JavaScript أيضًا. ما تفعله هذه الشيفرة بسيط جدًا، فهي تعمل على تغيير قيمة show بين true و false بشكل متناوب كلما نقر المستخدم على الزر.

وضعت الموجه `v-if` ضمن عنصر `p` الذي سيحتوي على النص `Welcome to Hsoub Academy`. أسندت لهذا الموجه قيمة الحقل `show`. فعندما تكون قيمة `show` تساوي `true` سيظهر عنصر `p` مع الرسالة المطلوبة للمستخدم، وإلا سيختفي عنصر `p` بشكل كامل، ليس من أمام المستخدم فحسب، وإنما من كامل بنية DOM، وهذا النقطة من الضروري الانتباه إليها.

من الممكن أيضًا استخدام الموجه `v-else` بعد الموجه `v-if`. يلعب الموجه `v-else` نفس الدور الذي تلعبه عبارة `else` في JavaScript، انظر إلى المثال البسيط التالي وهو تعديل بسيط عن المثال السابق، سيكون التعديل على شيفرة HTML فقط:

```
<div id="app">
  <button v-on:click="show = !show">
    Click this!
  </button>
  <p v-if="show">
    Welcome in Hsoub Academy!
  </p>
  <p v-else>
    Welcome in Vue.js!
  </p>
</div>
```

الإضافة الوحيدة هي:

```
<p v-else>
  Welcome in Vue.js!
</p>
```

استخدمت الموجه `v-else` وهكذا فعندما ينقر المستخدم الزر بشكل متكرر، ستتناوب قيمة `show` تبعًا لذلك بين `true` و `false` عندما تكون قيمة `show` تساوي `true` يظهر النص `Welcome to Hsoub Academy!` أما عندما تكون `false` يظهر النص `Welcome in Vue.js!`. ويمكن أيضًا اعتبارًا من الإصدار 2.1 للمكتبة Vue.js استخدام الموجه `v-else-if` الذي يقابل `else if` في لغة JavaScript.

يمكن استخدام كل من الموجهات الشرطية السابقة مع عناصر HTML أخرى مثل `div` و `template`. الميزة في استخدام العنصر `template` هو أنه لا يظهر في DOM عند عرض الصفحة. فهو يلعب دور حاوية لا تُصير عند عرض الصفحة. سأعدّل المثال السابق ضمن قسم HTML فقط:

```
<div id="app">
  <button v-on:click="show = !show">
```

```

    Click this!
  </button>
  <template v-if="show">
    <h2>
      Welcome to Hsoub Academy!
    </h2>
    <p>
      Here you can learn Vue.js.
    </p>
  </template>
</div>

```

وضعت العنصرين h1 و p ضمن العنصر template. هذه المرة، وضعت الموجه v-if ضمن العنصر template وهكذا ستمكن من إخفاء كامل العنصر template مع العناصر الموجودة ضمنه، أو إظهاره بعنصره، وذلك بحسب قيمة الحقل show كما مر معنا.

يمكن استخدام العنصر div مكان العنصر template بنفس الأسلوب تمامًا، ولكن في هذه الحالة يظهر العنصر div ضمن DOM في حال كان شرط الإظهار محققًا (قيمة الحقل show تساوي true) في حين لا يظهر العنصر template في DOM حتى ولو كان شرط الإظهار محققًا، فالأمر يعود إليك في تحديد العنصر المناسب لاحتياجاتك.

### 3.3 الفرق بين v-if و v-show

من الممكن استخدام الموجه v-show عوضًا عن الموجه v-if، في إخفاء أو إظهار عنصر وفقًا لشرط معين كما وسبق أن رأينا قبل قليل.

يكمن الفرق الأساسي بين الموجهين السابقين، في أن الموجه v-if يؤدي إلى إزالة العنصر بشكل كامل من DOM كما لو أنه غير موجود بالأصل. أما الموجه v-show فيعمل على إخفاء العنصر فقط، دون إزالته بالكامل من DOM.

يمكن أن نلاحظ الفرق بينهما بسهولة من خلال المثال البسيط التالي:

```

<div id="app">
  <button v-on:click="show = !show">
    Click this!
  </button>
  <p v-if="show">
    Welcome to Hsoub Academy!
  </p>
</div>

```

لاحظ أنني استخدمت الآن الموجه `v-if`. بالنسبة لقسم JavaScript فهو نفسه كما في الأمثلة السابقة. عندما يكون النص `Welcome to Hsoub Academy!` ظاهرًا. انقر عليه بزر الفأرة الأيمن واختر `Inspect` من القائمة المنبثقة إذا كنت تستخدم Google Chrome أو اختر `Inspect Element` إذا كنت تستخدم FireFox. ستري شيئًا شبيهًا بما يلي:

```

<button>
  Click this!
</button>
...
<p>
  Welcome in Hsoub Academy!
</p> == $0
</div>
▶ <script>...</script>
</body>

```

انقر الآن على الزر، سيختفي النص بالطبع. انظر إلى نافذة العناصر مرة أخرى، لتجد أن عنصر `p` الذي يحتوي النص السابق قد اختفى تمامًا من المستند، وحلّ مكانه رمز تعليق (`comment`) كما في الشكل التالي (لاحظ المستطيل الأحمر):

```

▼ <div id="app">
...
  <button>
    Click this!
  </button> == $0
  <!-->
</div>
▶ <script>...</script>

```

كرّر الآن نفس التجربة السابقة، ولكن بعد أن تستبدل بالموجه `v-if` الموجه `v-show`. هذه المرة لن يختفي العنصر `p` كليًا من DOM، إنما ستستخدم Vue.js تنسيق CSS واسمه `display` لإخفائه من أمام المستخدم دون إزالته تمامًا من DOM. انظر الشكل التالي (لاحظ المستطيل الأحمر مرة أخرى):

```

▼ <div id="app">
...
  <button>
    Click this!
  </button>
  <p style="display: none;">
    Welcome in Hsoub Academy!
  </p> == $0
</div>
▶ <script>...</script>

```

## 3.4 تصيير القوائم باستخدام v-for

من الممكن توليد قائمة من العناصر اعتبارًا من بيانات موجودة على شكل مصفوفة وذلك بشكل تلقائي من خلال استخدام الموجه v-for، وهو يشبه إلى حد كبير حلقة for التكرارية في لغات البرمجة عمومًا. ليكن لدينا مصفوفة اسمها fruits تحتوي على العناصر التالية: Apple و Banana و Orange و Kiwi. نريد إظهار هذه البيانات ضمن قائمة غير مرتبة. يمكن استخدام البرنامج التالي:

```
<div id="app">
  <ul>
    <li v-for='fruit in fruits'>{{fruit}}</li>
  </ul>
</div>
```

```
var app = new Vue({
  el: "#app",
  data: {
    fruits: ['Apple', 'Banana', 'Orange', 'Kiwi']
  }
})
```

وضعنا الموجه v-for ضمن العنصر li على النحو التالي:

```
v-for='fruit in fruits'
```

اسم المتغير fruit هنا كافي، يمكنك استخدام أي اسم آخر. أما fruits فهو نفس الحقل المعرف ضمن القسم data في كائن Vue.js. النتيجة ستكون على النحو التالي:

- Apple
- Banana
- Orange
- Kiwi

من الممكن بالطبع أن نحصل على البيانات الموجودة ضمن الحقل fruits من خدمة ويب (Web Service) مثلاً.

إذا أردت أن تحصل على دليل (index) العنصر أيضًا، يمكن ذلك بسهولة بإجراء التعديل التالي على الموجه v-for:



```
<li v-for='(fruit, i) in fruits'>{{fruit}} - ({{i}})</li>
```

الإضافتين الجديدتين: (fruit, i) و ({{i}}). لاحظ كيف وضعنا المتغير fruit أولاً ثم المتغير الذي سيعبر عن الدليل بعده، والاثنان ضمن قوسين عاديين (fruit, i) والترتيب بهذه الصورة مهم بالطبع. عند التنفيذ سيظهر دليل كل عنصر بجواره بين قوسين عاديين، مع الملاحظة أنّ الترقيم سيبدأ من الصفر، أي أنّ دليل العنصر الأول سيكون صفراً، وهذا بديهي بالطبع. في الواقع يذكرني هذا الأسلوب المتمثل في استخلاص الدليل، بأسلوب لغة البرمجة بايثون في ذلك.

## 3.5 المرور على خصائص كائن

يمكن استخدام الموجه v-for أيضاً في المرور على خصائص كائن ما. سيكون هذا الكائن عبارة عن كائن JavaScript عادي سنعمل باستخدام هذا الموجه على استخلاص قيمة كل خاصية مع اسمها. ليكن لدينا الكائن التالي:

```
customer: {name: 'Ahmad', age:30, items: 3}
```

يتألف هذا الكائن من ثلاث خصائص: name و age و items.

سنمر الآن على قيم هذه الخصائص على النحو التالي:

```
<div v-for='value in customer'>
  <p>
    {{value}}
  </p>
</div>
```

```
var app = new Vue({
  el: "#app",
  data: {
    customer: {name: 'Ahmad', age:30, items: 3}
  }
})
```

يمكنك بالطبع استخدام أي عنصر من HTML لعرض هذه القيم. ستظهر كل قيمة ضمن عنصر p خاص بها.

إذا أردنا تطوير المثال البسيط السابق، بحيث يصبح من الممكن أن يكون لدينا مجموعة من الزبائن (customers) ونريد أن نمرّ على هذه المجموعة، مع عرض خصائص كل كائن (زبون) منها، فإننا سنستخدم موجهي v-for متداخلين على النحو التالي:

```
<div id="app">
  <div v-for='customer in customers'>
    <div v-for='value in customer'>
      <p>
        {{value}}
      </p>
    </div>
  </div>
  <hr>
</div>
</div>
```

```
var app = new Vue({
  el: "#app",
  data: {
    customers: [{
      name: 'Ahmad',
      age: 30,
      items: 3
    },
    {
      name: 'Saeed',
      age: 28,
      items: 13
    },
    {
      name: 'Majd',
      age: 21,
      items: 12
    }
  ]
})
```

لاحظ كيف أجريت تعديلاً على اسم الحقل `customer` ليصبح `customers` ضمن القسم `data` من كائن `Vue.js`. لاحظ أيضاً موجهي `v-for` المتداخلين، يمرّ الخارجي على كل عنصر من عناصر المصفوفة `customers`، في حين يمرّ الداخلي على جميع قيم الخصائص الموجودة ضمن كائن `customer` محدّد. ستحصل عند التنفيذ على خرج شبيه بما يلي:

Ahmad

30

3

Saeed

28

13

Majd

21

12

في المثالين الأخيرين حصلنا على قيمة الخاصية دون اسمها. يمكن بأسلوب مشابه لعملية الحصول على دليل العنصر ضمن المصفوفة، الحصول على اسم الخاصية بالإضافة إلى قيمتها. أجرِ التعديل البسيط التالي على موجه v-for الداخلي الذي يمرّ على الخصائص ليصبح على النحو التالي:

```
<div v-for="(value, key) in customer">
```

والترتيب هنا مهم أيضًا.

لاحظ أن Vue.js تراقب بعض توابع المصفوفة customers التي تعاملنا معها في المثال السابق باستخدام الموجه v-for. هذه التوابع التي سأسردها الآن، تُغيّر في الحالة الداخلية للمصفوفة، وبالتالي تعمل Vue.js إلى إحداث تغيير فوري على الخرج يتوافق مع التغيير الذي حدث. هذه التوابع هي:

```
push()
pop()
shift()
unshift()
splice()
sort()
reverse()
```

جَرِّب أن تضيف إلى المثال الأخير زر، وأسند إليه الموجه v-on:click على النحو التالي:

```
<button v-on:click='customers.push({name: "Hasan", age:24,
items:15})'>
  Add new customer
</button>
```

بعد تنفيذ البرنامج ونقر هذا الزر، ستلاحظ أن الزبون الذي اسمه Hasan قد أُضيف إلى الخرج تلقائيًا رغم أننا أضفناه إلى المصفوفة customers فقط.

## 3.6 ختام الفصل

تعلمنا في هذا الفصل كيفية التعامل مع الموجهات الشرطية مثل `v-if` و `v-else` و `v-else-if` و `v-show` وكيفية استخدامها في تطبيقات Vue.js. كما ميزنا بين المجهين `v-if` و `v-show` وتعلمنا متى نستخدم كل منهما. كما تعرّفنا أيضًا في هذا الفصل على الموجه `v-for` الذي يلعب نفس الدور الذي تلعبه الحلقات التكرارية في لغات البرمجة بشكل عام، واستخدمناه للمرور على عناصر المصفوفات، بالإضافة إلى المرور على خصائص كائن محدد.

## 3.7 تمارين داعمة

### 3.7.1 تمرين 1

أجر تعديلًا على التطبيق الزبائن customers السابق. في هذه المرة يجب أن تظهر أسماء الزبائن فقط، ضمن عنصر القائمة المنسدلة `select` وذلك باستخدام `v-for` أيضًا.

### 3.7.2 تمرين 2

يُطلب في هذا التمرين، تطوير التطبيق الموجود في التمرين 2 من الفصل السابق. هذه المرة ستجعل الشيفرة أسهل بكثير بعد استخدام الموجه `v-for` بدلًا من الأسلوب الذي استخدمته هناك.

## 4. المزيد حول كائن Vue.js

سنتعلم في هذا الفصل:

- إنشاء أكثر من كائن Vue.js
- الوصول إلى عناصر HTML مباشرةً باستخدام `$refs`
- الخصائص المحسوبة ضمن كائن Vue.js
- الخصائص المراقبة ضمن كائن Vue.js
- تثبيت قالب جديد باستخدام `$mount()`
- فصل القالب عن عنصر HTML المُستهدف
- ما هو المكوّن (Component)؟

سنتعلم في هذا الفصل المزيد عن كائن Vue.js، حيث سنتعرف على كيفية إنشاء أكثر من كائن Vue.js وكيف نصل إلى خصائص كل كائن من خلال شيفرة JavaScript عادية. وسنتعرف على الخصائص المحسوبة والخصائص المراقبة، وسنعود أيضًا إلى القوالب، وكيف نثبت قالب ما إلى عنصر HTML بشكل برمجي، كما سنتعرف على المكوّنات والحاجة إليها، وسنبني تطبيق بسيط للغاية يعتمد على مكوّن يعمل على التحويل من واحدة الكيلو غرام إلى واحد الباوند.

## 4.1 إنشاء أكثر من كائن Vue.js

من الممكن إنشاء أكثر من كائن Vue.js ضمن نفس التطبيق. حيث يمكن ربط كل كائن بعنصر `div` كما كنا نفعل مسبقًا. يمكن لكل كائن أن يعمل بصورة مستقلة عن الكائن الآخر، وفي ذلك فائدة كبيرة، إذ يمكننا تقسيم واجهة المستخدم الرئيسية إلى أجزاء متعددة، كل منها ينفذ وظيفة محدّدة، بدون أن تتداخل مع بعضها.

في الحقيقة تقودنا هذه الميزة المهمة إلى بناء المكونات (Components). يلعب المكوّن دورًا مهمًا في تنظيم الشيفرة البرمجية وفي عملية إعادة الاستخدام للشيفرة من قبلك، أو من قبل أي شخص آخر. سنتحدث عن المكونات في Vue.js بشكل مبدئي في هذا الفصل.

لنأخذ الآن مثالًا بسيطًا يوضح كيفية إنشاء أكثر من كائن واحد بنفس الوقت:

```
<div id="app1">
  {{title}}
</div>

<div id="app2">
  {{title}}
</div>
```

```
var instance1 = new Vue({
  el: '#app1',
  data: {
    title: 'From first instance'
  }
})

var instance2 = new Vue({
  el: '#app2',
  data: {
    title: 'From second instance'
  }
})
```

الشيفرة السابقة سهلة ومباشرة. بدايةً عرّفت عنصري `div` أسندت للأول المعرّف `app1` وللثاني المعرّف `app2`. كل من العنصرين السابقين لا يحتوي إلا على الاستبدال النصي `{{title}}`.

بالنسبة لشيفرة JavaScript فالأمر بسيط أيضًا. فقد عرّفت كائني Vue.js وأسندت كل منهما إلى متغيرين منفصلين `instance1` و `instance2`. الكائنين متشابهين من حيث الشكل العام ولكنهما يختلفان بالقيمة النصية للخاصية `title` لكل منهما كما هو واضح.

عند تنفيذ التطبيق السابق ستحصل على شكل شبيه بما يلي:

From first instance  
From second instance

من الواضح ظهور نصين مختلفين من كائنين مختلفين، رغم أنه لكل منهما نفس اسم الخاصية `title`. يمكن بالطبع وجود أكثر من كائنين، ويمكن بديهياً أن يكون لكل كائن عتاده الخاص من الخصائص والدوال وغيرها.

توجد أيضاً ميزة مهمة لكائنات `Vue.js`، وهي إمكانية الوصول لأي كائن من شيفرة `JavaScript` عادية. انظر إلى المثال المعدل عن المثال السابق (التعديل سيكون على شيفرة `JavaScript` فقط):

```
var instance1 = new Vue({
  el: '#app1',
  data: {
    title: 'From first instance'
  }
})

var instance2 = new Vue({
  el: '#app2',
  data: {
    title: 'From second instance'
  }
})

instance1.$data.title='This text from outside!';
```

التعديل الوحيد الذي حدث هو في السطر الأخير. انظر كيف كتبت شيفرة `JavaScript` عادية للوصول إلى الخاصية `title` للكائن `instance1`. بعد التنفيذ ستحصل على شكل شبيه بما يلي:

This text from outside!  
From second instance

لاحظ معي كيف أصبح السطر الأول. استطعت تعديل النص من خارج الكائن. وهناك أمر آخر، لعلك قد انتبهت إلى الخاصية `$data`. في الحقيقة هو كائن يولده `Vue.js` بشكل تلقائي لكي يسمح للمبرمجين بالوصول إلى الخصائص الداخلية للقسم `data`.

هذا دليل واضح على أن `Vue.js` مدمجة بشكل ممتاز مع `JavaScript` وليست بديلاً عنها، إنما مكّلة لها. سنشاهد عدداً من هذه الكائنات والدوال المولدة بهذه الطريقة.

## 4.2 الوصول إلى عناصر HTML مباشرة

توجد أكثر من طريقة للوصول إلى عناصر HTML ضمن DOM. تضيف Vue.js طريقة أخرى لها باستخدام المفتاح `ref`. يسمح هذا المفتاح للمبرمج بالوصول إلى أي عنصر HTML بسهولة كبيرة. يمكن استخدام هذا الأسلوب بالشكل التالي:

```
<div id="app">
  <button v-on:click='changeText' ref='testButton'>
    Old Text
  </button>
</div>

var app = new Vue({
  el: '#app',
  methods: {
    changeText: function() {
      this.$refs.testButton.innerText = 'New Text!';
    }
  }
})
```

الجديد هنا هو وضع الكلمة `ref` كما لو أنها سمة ضمن العنصر `button` وإسناد القيمة `testButton` لها. الكلمة `ref` ليست سمة قياسية في HTML بالتأكيد، إنما هي كلمة تابعة لـ Vue.js. انظر الآن إلى شيفرة JavaScript وتحديدًا ضمن التابع `changeText` ستلاحظ السطر التالي:

```
this.$refs.testButton.innerText = 'New Text!';
```

الخاصية الجديدة هنا هي `$refs` وهي عبارة عن كائن JavaScript مولّد تلقائيًا. انتبه إلى وجود حرف `s` الخاص بالجمع آخر كلمة `$refs`، لأنّه من الممكن استخدام أكثر من كلمة `ref` مع عناصر HTML مختلفة. انظر أيضًا إلى الخاصية `testButton` وهي تحمل نفس الاسم الذي عيّناه ضمن الكلمة `key` في HTML. في الحقيقة إنّ `testButton` عبارة عن كائن JavaScript أيضًا يمثّل عنصر في HTML، ولذلك استطعنا استخدام الخاصية `innerText` منه.

عند تنفيذ التطبيق السابق. ستحصل على زر وحيد يحمل النص `Old Text`. بعد نقر الزر، ستحصل على النص الجديد `New Text!` كما هو متوقع.

في الواقع لا يُنصح بتعديل خصائص عناصر HTML بهذا الأسلوب، أنصح باستخدام هذا الأسلوب فقط لقراءة خصائص عناصر HTML في حال الحاجة.



## 4.3 الخصائص المحسوبة في Vue.js

سنتحدث في هذه الفقرة عن مفهوم الخصائص أو الخاصيات المحسوبة (Computed Properties)، والحاجة إليها عند بناء تطبيقات باستخدام Vue.js. تشبه بنية الخصائص المحسوبة البنية الخاصة بالتوابع، في أنهما عبارة عن دوال، مع فرق بسيط يتمثل في أن الخاصية المحسوبة يجب أن تُرجع قيمة ما.

تُعرّف الخصائص المحسوبة ضمن قسم جديد اسمه `computed` يُوضع على نفس مستوى القسمين `data` و `methods` أي على الشكل التالي:

```
var app = new Vue({
  el: "#app",
  data: {
    ...
  },
  computed: {
    ...
  },
  methods: {
    ...
  }
})
```

تلعب الخصائص المحسوبة دورًا مهمًا عندما يكون التطبيق كبيرًا نسبيًا، حيث تسمح بتنفيذ الشيفرة البرمجية بشكل محسّن (Optimized)، فهي تتجنب المعالجة غير الضرورية لأجزاء من الشيفرة في حال لم يطرأ تغيير ما عليها. لفهم الخصائص المحسوبة بشكل أفضل، سنأخذ تطبيق يقيم درجات الحرارة برسائل نصية بسيطة لفهم الحاجة إلى الخصائص المحسوبة:

```
<div id="app">
  <span>Current temperature:</span><input type='text' v-
model='temperature' />
  <span> - Clouds:</span>
  <select v-model='clouds'>
    <option>Yes</option>
    <option>No</option>
  </select>
  <p>
    <b>Result:</b> (Computed) {{evaluation_computed}} | (Method)
    {{evaluation_method()}}
  </p>
  <p>
    {{clouds == 'Yes'? 'With some clouds.': 'And the sky is clear.'}}
  </p>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    temperature: 30,
    clouds: 'Yes'
  },
  computed: {
    evaluation_computed: function() {
      console.log('Computed');
      return this.temperature > 35 ? 'Hot' : this.temperature < 20 ?
      'Cold' : 'Moderate';
    }
  },
  methods: {
    evaluation_method: function() {
      console.log('Method');
      return this.temperature > 35 ? 'Hot' : this.temperature < 20 ?
      'Cold' : 'Moderate';
    }
  }
})
```

بعد تنفيذ البرنامج السابق ستحصل على خرج شبيه بما يلي:

Current temperature:  - Clouds:

**Result:** (Computed) Moderate | (Method) Moderate

With some clouds.

بالنسبة لقسم HTML أعتقد أن الأمور واضحة، فقد عرّفت مربع نص، بالإضافة إلى عنصر قائمة منسدلة يحمل القيمتين Yes و No للإشارة إلى وجود غيوم في السماء أم لا. كل من العنصرين السابقين مربوطين ربطًا ثنائي الاتجاه باستخدام الموجه v-model بالحقلين الموافقين من القسم data في كائن Vue.js.

ثم عرّفت بعد ذلك عنصري p لعرض النتائج. عنصر p الأول مسؤول عن عرض تقييم الوضع الحالي للجو فيما إذا كان حارًا أم معتدلًا أم باردًا بحسب قيمة الحقل temperature، ولاحظ هنا أنني أعرض التقييم من الخاصية المحسوبة evaluation\_computed والتابع evaluation\_method() على التوالي لغرض سأوضحه لك بعد قليل. أما عنصر p الثاني فيستخدم لإخبار المستخدم في حال وجود بعض الغيوم أم أن السماء صافية بحسب القيمة التي اختارها المستخدم من عنصر القائمة المنسدلة السابق. لا أدري إن كنت قد انتبهت إلى أنني لا أضع قوسي الاستدعاء بعد اسم الخاصية المحسوبة على عكس التابع العادي المعرّف ضمن القسم methods.

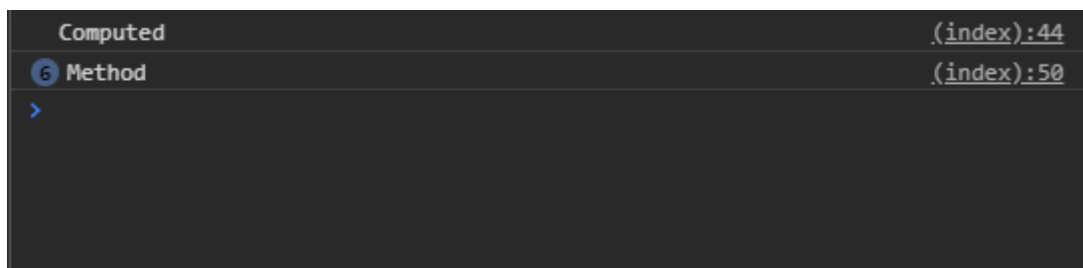
بالنسبة لشفيرة JavaScript، لاحظ بدايةً أنَّ الشيفرة البرمجية الموجودة سواءً في الخاصية المحسوبة `evaluation_computed` أم في التابع `evaluation_method()` متطابقة تمامًا. ولاحظ أيضًا أنني قد وضعت في كل منهما تعليمة الكتابة إلى الطرفية (`console`) الخاصة بأدوات المطور ضمن المتصفح لمراقبة كيف ستنفذ كل منهما.

اعرض أدوات المطور في المتصفح الآن (اضغط المفتاح F12)، ثم غير درجة الحرارة في مربع النص. ستلاحظ ظهور الكلمتين `Computed` و `Method` على التوالي، مما يشير إلى أنَّ التنفيذ قد دخل إلى الخاصية المحسوبة `evaluation_computed` والتابع `evaluation_method()`. ستستمر الكلمتان السابقتان بالظهور على هذا النمط، كلما أجريت أي تغيير في درجة الحرارة.

الأمر المثير الآن، هو أنه إذا غيرت الاختيار الحالي ضمن عنصر القائمة المنسدلة، ستجد الكلمة `Method` قد ظهرت وحدها ضمن الطرفية، ولن تظهر الكلمة `Computed` مما يشير إلى أنَّ الشيفرة البرمجية الموجودة ضمن الخاصية المحسوبة لم تُنفَّذ!

إدًا، فالخاصية المحسوبة ذكية بما يكفي لكي تدرك أنَّ درجة الحرارة لم تتغير فليس هناك حاجة لتنفيذ الشيفرة البرمجية الخاصة بها كل مرة. بعبارة أخرى، تدرك الخاصية المحسوبة أنَّ الشيفرة البرمجية الموجودة ضمنها لا تحتوي على خصائص تم تغييرها بشكل أو بآخر، فتختصر عملية التنفيذ وتعرض التقييم السابق لدرجة الحرارة طالما أنها لم تتغير!

النتيجة التالية نتج معي بعد تغيير حالة الغيوم عدة مرات دون تغيير قيمة درجة الحرارة سوى المرة الأولى فقط:



## 4.4 الخصائص المراقبة ضمن كائن Vue.js

رغم أنَّ الخصائص المحسوبة كافية في معظم الأحيان إلا أنه هناك بعض الحالات التي يكون استخدام الخصائص المراقبة أفضل، والخصائص المراقبة (`Watched Properties`) تشبه إلى حد كبير الخصائص المحسوبة، وتتميز عنها بميزتين أساسيتين:

1. يمكن تنفيذ شيفرة برمجية غير متزامنة ضمنها، بمعنى أنه يمكن تنفيذ مهام بشكل متوازي مع التطبيق الأساسي كالاتصال بخادوم بعيد مثلاً.

2. لا نحتاج إلى إرجاع قيمة من الخصائص المراقبة كما كنا نفعل مع الخصائص المحسوبة.

تعرّف الخصائص المحسوبة ضمن قسم جديد اسمه `watch` يُوضع على نفس مستوى باقي الأقسام الأخرى مثل `data` و `methods` و `computed` أي على الشكل التالي:

```
var app = new Vue({
  el: "#app",
  data: {
    ...
  },
  computed: {
    ...
  },
  watch: {
    ...
  },
  methods: {
    ...
  }
})
```

لنأخذ مثالاً بسيطاً:

```
<div id="app">
  <input type="text" v-model='content' />
  <p>
    The input has changed: {{counter}} times.
  </p>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    content: '',
    counter: 0
  },
  watch: {
    content: function() {
      tmp = this;
      setTimeout(function() {
        tmp.counter++;
      }, 2000);
    }
  }
})
```

```
}
})
```

استخدمت في المثال السابق مربع إدخال نصي يسمح للمستخدم بكتابة ما يرغب. عندما يكتب المستخدم أي شيء ضمن مربع النص، سيعمل التطبيق على إحصاء عدد مرّات التعديل التي أجراها المستخدم، ويظهر ذلك في الأسفل ضمن رسالة مناسبة.

استخدمت الربط ثنائي الاتجاه باستخدام v-model للربط مع الحقل content (راجع الفصل الثاني). سنراقب الحقل content (المرتبط مع دخل المستخدم) في حال حدث أي تغيير على قيمته. وهكذا فقد وضعت نفس اسم الحقل content ضمن القسم watch كما هو واضح، وأسندت إليه تابع. هذه هي الصيغة المعتمدة لمراقبة أي حقل.

الذي يحدث ضمن الخاصية المراقبة بسيط في هذا المثال. فقد استخدمت التابع الضمن setTimeout لزيادة قيمة الحقل counter بعد ثانيتين في حال حدث أي تغيير عليه. اعتمدت هذا الأسلوب، لكي ألفت انتباهك إلى أنّ عملية العد تحدث بشكل غير متزامن، وبشكل منفصل تمامًا عن التعديلات التي يجريها المستخدم ضمن مربع النص.

نقدّ التطبيق السابق، وابدأ بالكتابة والتعديل ضمن مربع النص. ستلاحظ أنّ قيمة العدّاد ستزيد بينما تجري التعديلات ضمن مربع النص، ولكن هذا التزايد لا يحدث فوراً، إنّما بتأخير زمني قدره ثانيتين عن التعديلات الفعلية.

لاحظ أنّني استخدمت المتغير المؤقت tmp لتخزين المرجع this قبل استخدام التابع setTimeout ضمن الخاصية المراقبة.

والسبب في ذلك أنّ الشيفرة البرمجية سننقذ ضمن مغلف (Closure) وبالتالي لن تشير الكلمة this إذا استخدمت مباشرةً إلى كائن Vue.js كما اعتدنا سابقاً.

## 4.5 تثبيت قالب جديد باستخدام \$mount()

استخدمنا في جميع الأمثلة التي تعاملنا معها حتى الآن القسم el من كائن Vue.js لتحديد العنصر المُستهدف الذي سيمثل القالب template الذي سيعمل التطبيق على تعديله والتعامل معه كما وسبق أن أوضحنا من قبل (راجع "فهم قوالب Vue.js" من الفصل الثاني).

إلا أنه يمكن الاستغناء عن القسم `el` بشكل كامل، إذا لم تكن نعرف مسبقًا العنصر الذي سنستهدفه (وبالتالي لا نعرف القالب). في هذه الحالة يمكن استخدام التابع المولّد تلقائيًا `$mount()` الذي ينوب عن القسم `el` في هذه الحالة، وذلك في اللحظة التي نريد فيها الربط مع عنصر محدّد. انظر إلى المثال البسيط التالي لتوضيح هذه الفكرة:

```
<div id="app">
  {{title}}
</div>
```

```
var app = new Vue({
  data: {
    title: 'Hello!'
  }
})
```

لاحظ أنني قد حذف القسم `el`، وهكذا، وعند تنفيذ المثال السابق ستحصل على الخرج التالي:

```
{{title}}
```

أي أنّ Vue.js لم يعرف في هذه الحالة القالب الذي سيتعامل معه. لحل هذه المشكلة وإظهار الرسالة المناسبة، سنضيف سطرًا واحدًا فقط إلى شيفرة JavaScript السابقة، لتصبح الشيفرة على النحو التالي:

```
var app = new Vue({
  data: {
    title: 'Hello!'
  }
});

app.$mount('#app');
```

إذاً فقد استخدمنا التابع `$mount()` ومزّنا له معزف العنصر المُستهدف (وهو `#app` في مثالنا) ليعمل عندها Vue.js على ربط (mount) القالب المراد التعامل معه، وبالتالي إظهار الرسالة المناسبة للمستخدم. تُعتبر هذه الميزة، من المزايا المهمة جدًا في Vue.js حيث أنّها تسمح ببناء المكونات التي سنتحدث عنها بعد قليل.

## 4.6 فصل القالب عن عنصر HTML المستهدف

توجد ميزة مهمة أخرى سنحتاج إليها لاحقًا عند العمل الموسّع مع المكونات، وهي إمكانية عدم كتابة شيفرة HTML التي (تعبّر عن القالب) ضمن العنصر المُستهدف. بمعنى آخر، يمكن فصل شيفرة HTML التي

تُعبّر عن القالب عن العنصر المُستهدف، ووضعها ضمن قسم جديد ضمن كائن Vue.js. اسم هذا القسم هو `template` ويوضع على نفس المستوى مع بقية الأقسام الرئيسية. انظر إلى الشكل التالي:

```
var app = new Vue({
  el: "#app",
  template: "HTML CODE GOES HERE",
  data: {
    ...
  },
  computed: {
    ...
  },
  watch: {
    ...
  },
  methods: {
    ...
  }
})
```

لنأخذ مثالاً بسيطاً يوضح هذه الفكرة:

```
<div id="app">

</div>
```

```
var app = new Vue({
  el: '#app',
  template: '<h2>Hsoub Academy</h2>'
});
```

بالنسبة لشفرة HTML فهي لا تحتوي سوى العنصر المُستهدف وهو فارغ بالطبع. أما بالنسبة لشفرة JavaScript فهي تحتوي على كائن Vue.js بسيط، عرّفنا ضمنه العنصر المُستهدف عن طريق `el`، وأيضاً وضعنا القالب الذي نريد التعامل معه ضمن القسم `template`. جعلت كود HTML في هذا القالب بسيطاً للغاية بهدف شرح الفكرة فقط.

نقدّ المثال السابق لتحصل على الجملة:

```
Hsoub Academy
```

يمكن أيضاً الاستغناء عن القسم `el` كلياً واستبداله بالتابع `$.mount()`. انظر لشفرة JavaScript الجديدة:

```
var app = new Vue({
  template: '<h2>Hsoub Academy</h2>'
});
```

```
});  
  
app.$mount('#app');
```

بعد التنفيذ، ستحصل على النتيجة السابقة. الآن لنجري تطويرًا بسيطًا على المثال الأخير، بحيث نعرض قيمة حقل اسمه `title` معزف ضمن القسم `data`. سيكون التعديل ضمن شيفرة JavaScript فقط:

```
var app = new Vue({  
  template: '<div><h2>Hsoub Academy</h2><p>{{title}}</p></div>',  
  data: {  
    title: 'Welcome dear user!'  
  }  
});  
  
app.$mount('#app');
```

الشفرة السابقة سهلة ومباشرة. بعد التنفيذ ستحصل على شكل شبيه بما يلي:

## Hsoub Academy

Welcome dear user!

لاحظ كيف حدث الاستبدال النصي ضمن شيفرة HTML الموجودة ضمن القالب، وذلك بسبب وجود `{{title}}`. هناك ملاحظة بسيطة أخرى حول شيفرة HTML المكتوبة ضمن القالب. يجب أن يحتوي القالب المُسنَد إلى القسم `template` على عنصر جذر واحد، في مثالنا السابق أنشأت عنصر `div` لهذا الغرض، ووضعت فيه العنصرين `h2` و `p` كما هو واضح.

مرّة أخرى يُعد هذا الأسلوب أساسيًا في بناء المكونات واستخدامها في Vue.js. في الحقيقة لا نستخدم هذا الأسلوب كما هو في التطبيقات العملية عادةً. الذي يهّمنا هنا هو المفهوم فقط.

## 4.7 ما هو المكون (Component)؟

المكون بصورة عامة عبارة عن وحدة برمجية مستقلة بذاتها، تُنجز عملاً واحدًا على الغالب. تصادف المكونات كثيرًا في عالم البرمجيات. وإذا أردت أمثلة عنها في عالم الويب، فجداول البيانات التي تعرض المعلومات المختلفة للمستخدم مع مزايا الترتيب، وأيضا المساحات الصغيرة الموجودة على جانب الصفحة التي تعرض درجة الحرارة الحالية أو أسعار الصرف للعملة، كلها عبارة عن مكونات.

وبصورة عامة أي ناحية وظيفية يمكن أن تُستخدم بشكل متكرر في نفس المشروع البرمجي أو في مشاريع برمجية مختلفة يمكن أن تُرشح لتصبح مكونًا.



سنلامس المكونات في هذا الفصل، ولن ندخل في تفاصيلها، حيث سنؤجل ذلك إلى فصول لاحقاً. سأبني في هذا الفصل مكون بسيط للغاية، وظيفته التحويل من واحدة الكيلوغرام إلى واحدة الباوند. انظر إلى التطبيق التالي:

```
<div id="app">
  <weightconverter></weightconverter>
  <weightconverter></weightconverter>
  <weightconverter></weightconverter>
</div>
```

```
var wcComponent = Vue.component('weightconverter', {
  template: `<div style='margin-bottom:10px;'>
    <input type='text' v-on:input='inputChanged' />
    <span>Kg. is equivalent to: <b>{{pounds}}</b> pounds.</span>
  </div>`,
  data: function() {
    return {
      pounds: 0
    }
  },
  methods: {
    inputChanged: function(event) {
      this.pounds = Number(event.target.value) * 2.20462;
    }
  }
});

// comment
var app = new Vue({
  el: '#app',
  components: {
    'weightconverter': wcComponent
  }
});
```

نفذ التطبيق السابق، ستحصل على شكل شبيه بما يلي:

<input type="text" value="10"/>	Kg. is equivalent to: <b>22.0462</b> pounds.
<input type="text" value="15"/>	Kg. is equivalent to: <b>33.0693</b> pounds.
<input type="text" value="25"/>	Kg. is equivalent to: <b>55.1155</b> pounds.

سأتحدث عن شيفرة HTML بعد قليل. لننظر الآن إلى شيفرة JavaScript. لنبدأ بالقسم الأول من هذه الشيفرة حيث سجلنا مكون جديد باستخدام التابع `Vue.component`:

```
var wcComponent = Vue.component('weightconverter', {
  template: `<div style='margin-bottom:10px;'>
    <input type='text' v-on:input='inputChanged' />
    <span>Kg. is equivalent to: <b>{{pounds}}</b> pounds.</span>
  </div>`,
  data: function() {
    return {
      pounds: 0
    }
  },
  methods: {
    inputChanged: function(event) {
      this.pounds = Number(event.target.value) * 2.20462;
    }
  }
});
```

كما هو واضح فإننا سنسند المكوّن الذي يُرجعه التابع `component()` إلى المتغيّر `wcComponent`. يقبل التابع `component()` وسيطين. الوسيط الأوّل هو اسم المكوّن المراد إنشاءه، وهو عبارة عن قيمة نصيّة، أمّا الوسيط الثاني فهو كائن آخر يحوي إعدادات المكوّن.

لو تمعنّت النظر بهذا الكائن، فستجد أنّه يطابق كائن `Vue.js` قياسي مع اختلاف واحد بسيط. يكمن الاختلاف في القسم `data`. في كائنات `Vue.js` التي أنشأناها حتى الآن كُنّا نُسند للقسم `data` كائن عادي يحتوي على الحقول المراد التعامل معها ضمن التطبيق. أمّا في الشيفرة السابقة فيجب تعريف القسم `data` على أنّه تابع يُرجع الكائن الذي يحتوي الحقول المراد التعامل معها:

```
data: function() {
  return {
    pounds: 0
  }
}
```

ما عدا ذلك تبقى الأمور كما هي!

أمّا القسم الثاني من شيفرة `JavaScript` فتحتوي على تعريف كائن `Vue.js` عادي، ولكن مع وجود قسم جديد وهو القسم `components`. هذا القسم يحتوي على أيّة مكونات سيستخدمها التطبيق، وهي في حالتنا هذه المكوّن `weightconverter` الذي عزفناه قبل قليل:

```
var app = new Vue({
  el: '#app',
  components: {
    'weightconverter': wcComponent
  }
});
```

لاحظ كيف نسجل المكونات التي نريد استخدامها: اسم المكوّن يليه المرجع له (موجود ضمن المتغيّر `wcComponent` بالطبع).

بالنسبة إلى شيفرة HTML فهي بسيطة، حيث نستخدم العنصر الجديد `weightconverter` ضمن شيفرة HTML كما لو أصبح عنصر HTML نظامي. استخدمت هذا العنصر ثلاث مرّات من باب توضيح أنّه يمكنك استخدامه في أكثر من مكان ضمن الصفحة.

## 4.8 ختام الفصل

لقد كان هذا الفصل غنيًا بالمعلومات المتنوّعة والمهمّة خصوصًا لما يليه من فصول. سأحيلك بشكل متكرّر إلى هذا الفصل في المستقبل، كلّما اقتضت الضرورة إلى الرجوع إلى مفهوم ذكر هنا. سننتقل في الفصل التالي إلى مستوى جديد، حيث سنتعلّم كيفية بناء تطبيقات عملية حقيقية، وسنتوسّع بمفهوم المكوّنات.

## 4.9 تمارين داعمة

### 4.9.1 تمرين 1

أجر تعديلًا على تطبيق تحويل الوحدات الذي بنيناه قبل قليل، بحيث يسمح بالتحويل بالاتجاهين: من كيلوغرام إلى باوند، ومن باوند إلى كيلو غرام بشكل مباشر.

### 4.9.2 تمرين 2

أنشئ مكوّنًا جديدًا، عبارة عن مؤقت زمني تنازلي (Countdown Timer). قيمته الابتدائية دقيقة واحدة، ثم يتناقص حتى يصل للصفر. ثم استخدم هذا المكوّن ضمن تطبيق بسيط لتجربته فقط.

## 5. مدخل إلى التعامل مع المكونات

سنتعلم في هذا الفصل:

- تجهيز هيكل التطبيق على حاسوب محلي.
- بناء مكون جديد: مكون المهام.
- تحسين تجربة الاستخدام للمكون.
- تمرير وسائط إلى المكونات.
- إنشاء أكثر من نسخة من المكون ضمن نفس الصفحة.
- إضافة ميزة التصفية لمكون المهام.
- إضافة ميزة مهمة جديدة لمكون المهام.

سنتابع في هذا الفصل التعامل مع المكونات، حيث سنتعلم كيف نبني المكونات بصورة عملية. سنبنّي في هذا الفصل مكونًا بسيطًا لكنّه عملي، وهو مكون إدارة مهام مبسّط. الهدف من هذا التطبيق هو التعرّف على أسس بناء المكونات بشكل جيّد. سيعمل هذا التطبيق البسيط على السماح للمستخدم بعرض بعض المهام التي ينوي تنفيذها فيما بعد، مع إمكانية تحديد فيما إذا كان قد أنجز المهام أم لا باستخدام زر اختيار بسيط. وسنضيف إليه ميزة ترشيح بسيطة لإخفاء أو إظهار المهام المنجزة، بالإضافة إلى إمكانية إضافة مهام جديدة.

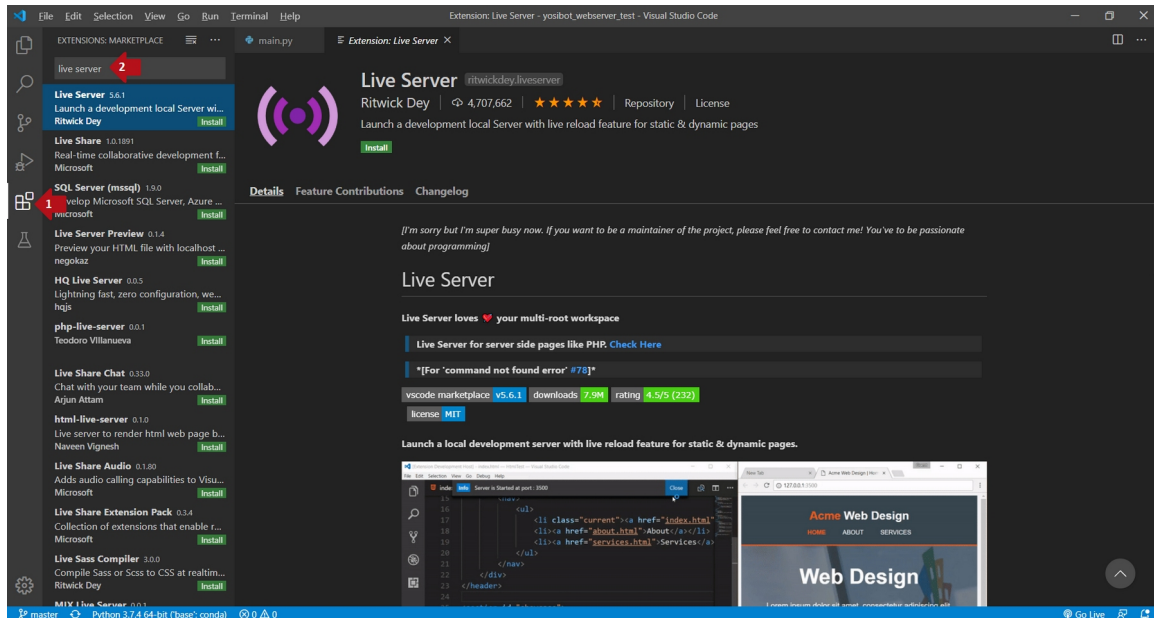
## 5.1 تجهيز هيكل التطبيق على حاسوب محلي

سنسلك هذه المرة منحى مغايرًا عما اعتدناه في الفصول السابقة. كان تركيزنا في الفصول السابقة منصبًا على كتابة تطبيقات vue.js ضمن موقع JSFiddle، وهذا الأمر جيّد في الواقع عندما نريد التعلّم أو تجريب بعض المزايا الخفيفة. ولكن عند الانتقال إلى مستوى أعلى ببناء تطبيقات عملية وواقعية، ينبغي علينا بالتأكيد الانتقال إلى أدوات تطوير فعّالة ومناسبة.

تشتمل هذه الأدوات بطبيعة الحال على خادوم نستخدمه أثناء بناء التطبيق، نحاكي من خلاله سلوك الخواديم الفعلية التي ستستضيف تطبيقنا النهائي الذي سيعمل عليه المستخدم في نهاية المطاف. في الحقيقة، أنصح دومًا بجعل ظروف تجريب التطبيق مماثلة قدر المستطاع لما سيكون عليه الوضع النهائي للتطبيق.

سنستخدم في هذا الفصل وغيره من الفصول اللاحقة، محرّر الشيفرة البرمجية Visual Studio Code من Microsoft. يمكنك في الواقع اختيار المحرّر الذي ترغب به، أو حتى يمكنك استخدام بيئة تطوير متكاملة إن أحببت. يوجد العديد من محرّرات النصوص البرمجية الأخرى مثل Atom و Sublime Text و Brackets. لتنصيب Visual Studio Code يمكنك زيارة الصفحة التالية [code.visualstudio.com/download](https://code.visualstudio.com/download). يمكنك اختيار نظام التشغيل المناسب لك من الأسفل. بالنسبة لنا سنختار النسخة الخاصة بويندوز. بعد التنزيل، نصّب التطبيق مع ترك الخيارات الافتراضية كما هي (قد تحتاج إلى صلاحيات مدير النظام).

بعد تثبيت Visual Studio Code انتقل إلى الإضافات Extensions الخاصة به، واعمل على تثبيت الإضافة Live Server التي سنستخدمها كخادوم بسيط. يمكنك الوصول مباشرةً إلى مدير الإضافات من الناحية اليسرى من الشاشة، كما هو ظاهر من الشكل التالي. بعد ذلك أدخل اسم الإضافة: Live Server في خانة البحث. بعد أن يجده، اختره، لتظهر النافذة الخاصة به كما في الشكل السابق، ثم انقر الزر الأخضر Install لتثبيته. انظر الشكل التالي:



من المفيد أيضًا تثبيت الإضافتين التاليتين:

- الإضافة Vetur لتنسيق الشيفرة الخاصة بـ .vue.js.

- الإضافة HTML5 Boilerplate لتنسيق شيفرة HTML.

اكتب اسم كل من هاتين الإضافتين في خانة البحث، واعمل على تثبيتهما كما فعلنا قبل قليل مع

الإضافة Live Server.

أنصح بإعادة تشغيل Visual Studio Code عند هذه المرحلة حتى ولو لم يطلب منك ذلك.

لنبدأ ببناء مشروعنا! انتقل إلى المكان الذي ترغب فيه بإنشاء المشروع على القرص الصلب، وأنشئ مجلدًا

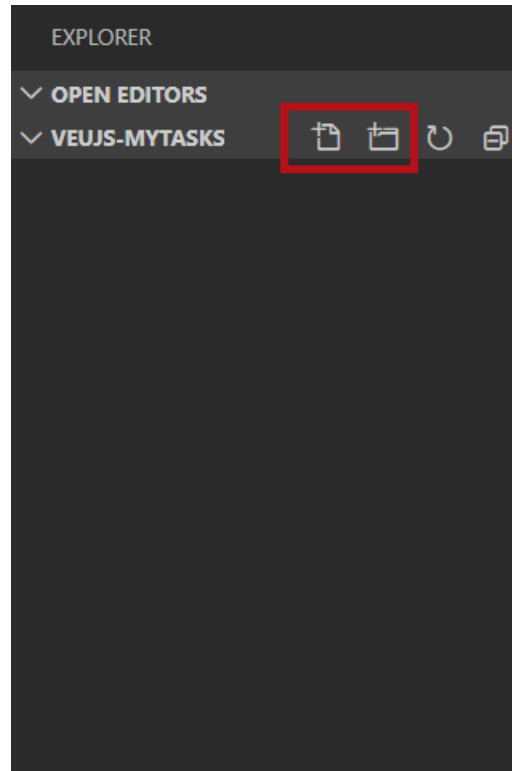
سمّه vuejs-mytasks. انتقل مرةً أخرى إلى Visual Studio Code ثم اختر الأمر Open Folder <- File

واختر المجلد الذي أنشأته تَوًّا.

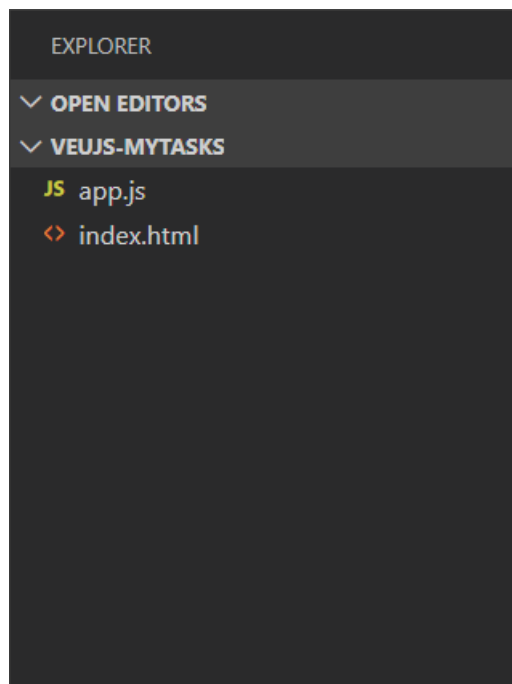
من نافذة المستكشف Explorer الموجودة في الطرف الأيسر، لاحظ الأيقونتين الصغيرتين، ضع مؤشر

الفأرة للحظات فوق كل منهما لتكتشف وظيفة كل منهما. الأيقونة الأولى من اليسار وظيفتها إنشاء ملف جديد

ضمن المجلد الحالي، والأيقونة الأخرى وظيفتها إنشاء مجلد جديد ضمن المجلد الحالي.



استخدم زر ملف جديد لثنشئ ملفين، واختر الاسمين index.html و app.js لهما على الترتيب. ستحصل في النهاية على شكل شبيه بما يلي:



اختر الملف index.html لكي تفتحه، ثم انسخ شيفرة HTML التالية إليه:

```

<!DOCTYPE html>
<html>

<head>
  <meta charset='utf-8'>
  <meta http-equiv='X-UA-Compatible' content='IE=edge'>
  <title>My Tasks</title>
  <meta name='viewport' content='width=device-width, initial-
scale=1'>
</head>

<body>
  <h1>Welcome to MyTasks Application</h1>
  <p>This application is built to explain how to deal with
components</p>

  <div id='app'>
    <tasks></tasks>
  </div>

  <script src="https://unpkg.com/vue@2.6.11/dist/vue.js"></script>
  <script src="app.js"></script>
</body>

</html>

```

شيفرة HTML السابقة عبارة عن شيفرة بسيطة، الأمر الجديد الوحيد فيها هو إضافة العنصر `tasks` وهو المكوّن الذي سنبنيه بعد قليل وسيمثّل قائمة المهام التي نرغب بنائها. لاحظ أيضًا أننا وضعنا هذا العنصر الجديد ضمن عنصر `div` له الوسم `id = 'app'` وهو العنصر المستهدف في كائن `vue.js` كما اعتدنا سابقًا. وأخيرًا، لاحظ كيف أضفت مرجعين لملف إطار العمل `vue.js` بالإضافة إلى مرجع للملف `app.js` الذي سيحتوي على الشيفرة البرمجية لكل من المكوّن والتطبيق:

```

<script src="https://unpkg.com/vue@2.6.11/dist/vue.js"></script>
<script src="app.js"></script>

```

لنتنقل الآن إلى الملف `app.js`، انسخ الشيفرة البرمجية التالية إليه:

```

Vue.component('tasks', {
  template: '<strong><p>{{name}} - Tasks</p></strong>',
  data() {
    return {
      name: 'Husam'
    }
  }
})

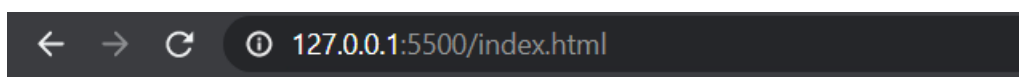
```



```
new Vue({
  el: '#app'
})
```

الشفرة البرمجية هنا مماثلة لتلك التي تعاملها معها في الفصل السابق، حيث نسجل مكون جديد باسم `tasks` بحيث نُسند له قالب بسيط، يعرض اسم الشخص الذي سُنسد إليه هذه المهام عن طريق الخاصية `{{name}}` كما هو واضح. ثم ننشئ كائن `vue.js` بسيط ونعيّن العنصر المستهدف.

انتقل الآن إلى نافذة المستكشف Explorer، وانقر بزر الفأرة الأيمن على الملف `index.html` ثم اختر الأمر `Open with Live Server` (تذكر أننا ثبتنا الإضافة `Live Server` قبل قليل)، سيؤدي ذلك إلى فتح نافذة أو تبويب جديد ضمن متصفح الانترنت الافتراضي لديك بحيث يتجه إلى العنوان `http://127.0.0.1:5500/index.html` وهو العنوان مع المنفذ الافتراضي الذي ينصت عنده الخادوم `Live Server`. ستحصل على شكل شبيه بما يلي:



## Welcome to MyTasks Application

This application is built to explain how to deal with components

**Husam - Tasks**

هذا دليل على أنَّ الأمور تسير على ما يرام. وأنَّ نجحنا ببناء الهيكل العام للتطبيق. لننتقل الآن للمرحلة التالية.

## 5.2 بناء مكون جديد: مكون المهام

لنبدأ الآن بالعمل الفعلي في بناء المكون الخاص بالمهام، والذي أسميناه `tasks`. سأنقل أولاً شيفرة HTML المسندة للحقل `template` ضمن المكون، وأضعها ضمن مكان منفصل لأنها ستصبح بعد قليل كبيرة ومعقدة بعض الشيء لثوضع في مكان كهذا. أجرِ التعديل التالي في الملف `app.js` ضمن الحقل `template` للمكون ليصبح على النحو التالي:

```
template: '#tasks-template'
```

لاحظ أنني قد عرضت مكان التعديل فقط طلبًا للاختصار. الجديد هنا أنني وضعت معرّف القالب الجديد الذي سيحتوي على الشيفرة. انتقل الآن إلى الملف `index.html` وأضف الشيفرة التالية مباشرةً بعد عنصر `div` المُستهدف الخاص بتطبيق `vue.js`:

```
<script type='text/x-template' id='tasks-template'>
  <div>
    <h3>{{ name }} - Tasks</h3>
  </div>
</script>
```

كما ترى أجريت بعض التعديل على شيفرة HTML التي كانت موجودة سابقًا. انتقل الآن إلى الصفحة `index.html` في المتصفح ثم حدثها (إن لم تُحدث بشكل تلقائي)، يجب أن تحصل على شكل قريب من الشكل الذي حصلنا عليه في الفقرة السابقة.

الجديد هنا هو فصل القالب ووضعها ضمن مكان مخصص له. في هذه الحالة سيكون ضمن العنصر `script` والذي تحمل السمة `type` له القيمة `text/x-template` كما هو واضح. بالإضافة لذلك لاحظ كيف جعلت قيمة المعرف `id` له نفس القيمة التي أسندتها للحقل `template` ضمن المكوّن قبل قليل. من المهم جدًا تحقيق مبدأ الفصل في بناء المكونات. لأنه كلما أصبح المكوّن أكثر تعقيدًا كما ستري بعد قليل، كلما برزت الحاجة إلى تحقيق مبدأ الفصل بشكل أفضل.

لنكسب الآن مكوّننا الوليد بعض المزايا الإضافية لكي يصبح قادرًا على عرض بعض المهام للمستخدم. أجر التعديلات التالية ضمن الملف `app.js` ليصبح مشابهاً لما يلي:

```
Vue.component('tasks', {
  template: '#tasks-template',
  data() {
    return {
      name: 'Husam',
      tasks_list: [
        { title: "Write an introduction about vue.js", done: true },
        { title: "Drink a cup of team.", done: false },
        { title: "Call Jamil.", done: false },
        { title: "Buy new book.", done: true }
      ]
    }
  }
})
new Vue({
  el: '#app'
```

في الواقع، لقد أضفت حقلاً جديداً أسميته `tasks_list` يحتوي على بيانات المهام التي أرغب بعرضها. لاحظ كيف أنّ كل مهمة ضمن القائمة عبارة عن كائن بسيط، يحتوي على خاصيتين: تضم الأولى النص الخاص بالمهمة، أما الثانية فتضم قيمة منطقية تشير إلى أنّ المهمة قد نُفذت `true` أم ليس بعد `false`.

انتقل الآن إلى الملف `index.html` وأجر بعض التعديلات التي ستكون أكبر هذه المرة، ليصبح مماثلاً

لما يلي:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset='utf-8'>
  <meta http-equiv='X-UA-Compatible' content='IE=edge'>
  <title>My Tasks</title>
  <meta name='viewport' content='width=device-width, initial-
scale=1'>

  <style>
    .tasks-container {
      border-width: 1px;
      border-style: solid;
      display: inline-block;
margin-right: 20px;
      padding: 8px;
    }

    .w3-table-all {
      border-collapse: collapse;
      border-spacing: 0;
      width: 100%;
      display: table;
      border: 1px solid #ccc
    }

    .w3-table-all tr {
      border-bottom: 1px solid #ddd
    }

    .w3-table-all tr:nth-child(odd) {
      background-color: #fff
    }

    .w3-table-all tr:nth-child(even) {
      background-color: #f1f1f1
    }

    .w3-table-all td,
    .w3-table-all th {
```

```

        padding: 8px 8px;
        display: table-cell;
        text-align: left;
        vertical-align: top
    }

    .w3-table-all th:first-child,
    .w3-table-all td:first-child {
        padding-left: 16px
    }

    .w3-table-all th{
        background-color: #d0d0d0;
    }
</style>
</head>

<body>
    <h1>Welcome to MyTasks Application</h1>
    <p>This application is built to explain how to deal with
    components</p>

    <div id='app'>
        <tasks></tasks>
    </div>

    <script type='text/x-template' id='tasks-template'>
        <div class='tasks-container'>
            <table class='w3-table-all'>
                <colgroup>
                    <col style="width:15%">
                    <col style="width:85%">
                </colgroup>
                <tbody>
                    <tr>
                        <th colspan="2">
                            <center>{{ name }} - Tasks</center>
                        </th>
                    </tr>
                    <tr>
                        <td>
                            <strong>Done</strong>
                        </td>
                        <td>
                            <strong>Title</strong>
                        </td>
                    </tr>

                    <tr v-for="task in tasks_list" v-
bind:key="task.title">
                        <td>
                            {{ task.done }}

```

```

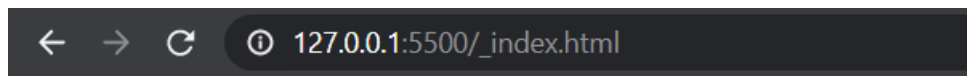
        </td>
        <td>
            {{ task.title }}
        </td>
    </tr>
</tbody>
</table>
</div>
</script>

<script src="https://unpkg.com/vue@2.6.11/dist/vue.js"></script>
<script src="app.js"></script>
</body>

</html>

```

حدّث الصفحة index.html ضمن متصفح الإنترنت لديك لتحصل على شكل شبيه بما يلي:



## Welcome to MyTasks Application

This application is built to explain how to deal with components

Husam - Tasks	
Done	Title
true	Write an introduction about VueJS components.
false	Drink a cup of team.
false	Call Jamil.
true	Buy new book.

ربما تشعر أنّ الشيفرة قد أصبحت كبيرة ومعقدة نسبيًا، إلّا أنّ الأمر في الحقيقة ليس كذلك. حدثت نسبة كبيرة من التعديلات عندما أضفت عنصر التنسيق `style` مع تنسيقاته إلى الملف `index.html`. كان من الأفضل وضع تنسيقات CSS ضمن ملف مستقل، وهذا ما سأعمل عليه بعد قليل. التنسيقات المستخدمة هنا استعرتها من موقع [W3Schools](https://www.w3schools.com) الشهير. بالإضافة إلى ذلك، لاحظ أنّني قد استخدمت عنصر الجدول `table` لعرض قائمة المهام. عدا عن ذلك، أعتقد أنّ معظم الشيفرة البرمجية واضحة، باستثناء الموجه الجديد: `v-bind:key`

```
<li v-for="task in tasks_list" v-bind:key="task.title">
```

في الواقع لقد استخدمنا موجه آخر مسبقًا، وهو `v-bind:href` وذلك في الفصل الثاني (استخدام `vue.js` للتعامل مع DOM). لكننا سنستخدم اليوم الكلمة `key` بدلاً من `href`. لاستخدام `v-bind:key` مزية مهمة تتمثل في الأداء (Performance)، وخصوصًا عندما يكون حجم البيانات كبيرًا. نستخدم هذا الموجه لتعيين مفتاح ربط `key` للموجه التكراري `for`.

من الواضح أنه في الوضع الحالي، من غير الممكن لنا أن نعدّل على أي مهمة بحيث تصبح منفذة أو غير منفذة. سنعمل في الفقرة التالية على تحسين تجربة الاستخدام، بتوفير إمكانية إجراء هذه التعديلات.

## 5.3 تحسين تجربة الاستخدام للمكون

سنعمل الآن على السماح للمستخدم بتعديل حالة المهمة وذلك بإضافة [عنصر اختيار Checkbox](#). قبل ذلك دعنا ننقل تنسيقات CSS إلى ملف منفصل. أنشئ ملف جديد ضمن نافذة المستكشف في Visual Studio Code لهذا الغرض ولنسمه `tasks.css`.

انقل محتويات العنصر `style` في الملف `index.html` إلى ملفنا الجديد، ثم احذف العنصر `style`. لاستخدام التنسيقات ضمن الملف الجديد، أضف مرجعًا إليه في الملف `index.html` ضمن القسم `head` على النحو التالي:

```
<link rel="stylesheet" href="tasks.css">
```

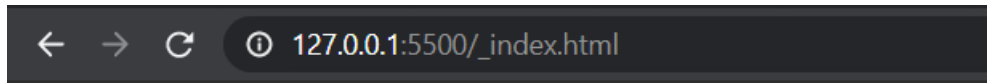
سنجري الآن تغييرًا ضمن الشيفرة البرمجية للقالب فقط، وتحديدًا بالقسم الخاص بإنهاء أو عدم إنهاء المهمة أي في القسم `Done` فقط. استبدل بالشيفرة التالي:

```
...
<td>
    {{ task.done }}
</td>
...
```

الشيفرة الجديدة التالية:

```
...
<td>
    <input type="checkbox" v-model="task.done"/>
</td>
...
```

أعد تحديث الصفحة لتحصل على شكل شبيه بما يلي:



## Welcome to MyTasks Application

This application is built to explain how to deal with components

Husam - Tasks	
Done	Title
<input checked="" type="checkbox"/>	Write an introduction about VueJS components.
<input type="checkbox"/>	Drink a cup of team.
<input type="checkbox"/>	Call Jamil.
<input checked="" type="checkbox"/>	Buy new book.

لاحظ أنني قد استخدمت الربط ثنائي الاتجاه `v-model="task.done"` للربط الثنائي للبيانات.

### الاختصارات في vue.js

يمكن دومًا استبدال الرمز `@` بالموّجّه `v-on`. أي أنّ `v-on:click` مثلًا سيصبح `:click`. وبنفس الأسلوب، يمكننا حذف الموّجّه `v-bind` بالكامل، وسيفهم `vue.js` أنّ هذا الموّجّه موجود. فمثلًا يمكن كتابة `href`: فقط بدلًا من `v-bind:href`، وكتابة `key`: بدلًا من `v-bind:key`.

## 5.4 تمرير وسائط إلى المكونات

لعلك قد لاحظت أننا استخدمنا بيانات ثابتة في التعامل مع مكّون المهام. ولكن في الواقع العملي سنحتاج إلى أن تكون هذه البيانات قابلة للتغيير كما هو واضح، لهذا الهدف توفّر المكونات في `vue.js` ميزة الخصائص `props`، حيث يمكن وضع قسم جديد اسمه `props` ضمن الوسيط الممرّر إلى المكّون عند إنشائه لهذه الغاية. في الحقيقة للخصائص الموجودة ضمن القسم `props` فوائد مهمة عندما سنتحدث عن المكونات المتداخلة لاحقًا في الفصل التالي، حيث سنستخدمها لتمرير البيانات من المكّون الأساسي إلى المكّون الفرعي.

لنبدأ بإجراء التعديلات اللازمة. أجر التعديلات التالية في الملف `app.js` ليصبح على الشكل التالي:

```
Vue.component('tasks', {
  template: '#tasks-template',
  props: {
    name: String,
    tasks_list: Array
  }
})

new Vue({
  el: '#app'
})
```

لقد أزلت قسم `data` وأضفت بدلاً منه القسم `props` (يمكن بالطبع أن يكونا معًا بنفس الوقت). عرفت الوسائط التي يمكن أن أمزرها للمكون ضمن القسم `props` بالشكل التالي:

```
props: {
  name: String,
  tasks_list: Array
}
```

الوسيط الأول هو `name` وهو من النوع `String` أي نص، والوسيط الثاني هو `tasks_list` وهو من نوع `Array` أي مصفوفة كما هو واضح. بالنسبة لطريقة الاستخدام فهي سهلة جداً. أجر التعديل التالي على الوسم `tasks` ضمن الملف `index.html`:

```
<tasks name='Husam' v-bind:tasks_list='[{ title: "Write an
introduction about vue.js components.", done: true },
{ title: "Drink a cup of team.", done: false },
{ title: "Call Jamil.", done: false },
{ title: "Buy new book.", done: true }]'></tasks>
```

مَرَرَت البيانات إلى المكون كما لو أنها وسوم عادية. الشيء الوحيد الملفت للنظر هو أنني قد استخدمت الموجه `v-bind` عند تمرير المصفوفة `tasks_list` وهذا الأمر إلزامي عند تمرير وسائط ديناميكية إلى المكونات، في حين أنه لا يجب استخدام هذا الموجه عند تمرير وسائط نصية ساكنة كما فعلنا عند تمرير الوسيط `name`.

لاحظ أنه يوجد العديد من أنواع الوسائط التي يمكن تمريرها إلى المكونات وهي:

```
String, Number, Boolean, Array, Object, Function, Promise
```



## 5.5 إنشاء أكثر من نسخة من المكون ضمن نفس الصفحة

لنعمل الآن على استخدام أكثر من مكون واحد ضمن الصفحة لنكتشف قوة المكونات. انسخ شيفرة HTML الأخيرة والصقها مرة أخرى مع إجراء بعض التعديلات البسيطة عليها:

```
<tasks name='My house' v-bind:tasks_list='[{ title: "Do a cleaning for windows.", done: false},
  { title: "Bring some vegetables and fruits.", done: true},
  { title: "Wash clothes", done: false}]'></tasks>
```

حدث الصفحة index.html من جديد لتحصل على شكل شبيه بما يلي:

### Welcome to MyTasks Application

This application is built to explain how to deal with components

Husam - Tasks	
Done	Title
<input checked="" type="checkbox"/>	Write an introduction about VueJS components.
<input type="checkbox"/>	Drink a cup of tea.
<input type="checkbox"/>	Call Jamil.
<input checked="" type="checkbox"/>	Buy a new book.

My house - Tasks	
Done	Title
<input type="checkbox"/>	Clean windows.
<input checked="" type="checkbox"/>	Bring some vegetables and fruits.
<input type="checkbox"/>	Wash clothes

يمكننا بسهولة أن نستخدم هذا المكون في المكان الذي نحتاج إليه في الصفحة، كما يمكن مشاركته مع الآخرين. مازال هناك الكثير للتحديث عنه حول المكونات، سنتناول في الفصول اللاحقة العديد من المزايا الخاصة بها، والطرق المثالية للتعامل معها. يمكنك الحصول على الشيفرات الكاملة بالملفات الثلاثة التي عملنا عليها في هذا الفصل في المقاطع الثلاثة التالية:

- الملف index.html:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset='utf-8'>
  <meta http-equiv='X-UA-Compatible' content='IE=edge'>
  <title>My Tasks</title>
  <meta name='viewport' content='width=device-width, initial-scale=1'>
  <link rel="stylesheet" href="tasks.css">
</head>

<body>
```

```

<h1>Welcome to MyTasks Application</h1>
<p>This application is built to explain how to deal with
components</p>

<div id='app'>
  <tasks name='Husam' v-bind:tasks_list='[{ title: "Write an
introduction about vue.js components.", done: true },
  { title: "Drink a cup of tea.", done: false },
  { title: "Call Jamil.", done: false },
  { title: "Buy a new book.", done: true }]'></tasks>

  <tasks name='My house' v-bind:tasks_list='[{ title: "Clean
windows.", done: false},
  { title: "Bring some vegetables and fruits.", done: true},
  { title: "Wash clothes", done: false}]'></tasks>
</div>

<script type='text/x-template' id='tasks-template'>
  <div class='tasks-container'>
    <table class='w3-table-all'>
      <colgroup>
        <col style="width:15%">
        <col style="width:85%">
      </colgroup>
      <tbody>
        <tr>
          <th colspan="2">
            <center>{{ name }} - Tasks</center>
          </th>
        </tr>
        <tr>
          <td>
            <strong>Done</strong>
          </td>
          <td>
            <strong>Title</strong>
          </td>
        </tr>
        <tr v-for="task in tasks_list" v-
bind:key="task.title">
          <td>
            <input type="checkbox" v-
model="task.done"/>
          </td>
          <td>
            {{ task.title }}
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</script>

```

```

        </table>
      </div>
    </script>

    <script src="https://unpkg.com/vue@2.6.11/dist/vue.js"></script>
    <script src="app.js"></script>
  </body>

</html>

```

• الملف tasks.css:

```

.tasks-container {
  border-width: 1px;
  border-style: solid;
  display: inline-block;
  margin-right: 20px;
  padding: 8px;
}

.w3-table-all {
  border-collapse: collapse;
  border-spacing: 0;
  width: 100%;
  display: table;
  border: 1px solid #ccc
}

.w3-table-all tr {
  border-bottom: 1px solid #ddd
}

.w3-table-all tr:nth-child(odd) {
  background-color: #fff
}

.w3-table-all tr:nth-child(even) {
  background-color: #f1f1f1
}

.w3-table-all td,
.w3-table-all th {
  padding: 8px 8px;
  display: table-cell;
  text-align: left;
  vertical-align: top
}

.w3-table-all th:first-child,
.w3-table-all td:first-child {
  padding-left: 16px
}

```

```

}

.w3-table-all th{
  background-color: #d0d0d0;
}

```

• الملف app.js:

```

Vue.component('tasks', {
  template: '#tasks-template',
  props: {
    name: String,
    tasks_list: Array
  }
})

new Vue({
  el: '#app'
})

```

## 5.6 إضافة ميزة الترشيح لمكون المهام

كثيرًا ما ستحتاج في التطبيقات التي ستكتبها إلى ميزة الترشيح بصرف النظر عن نوع التطبيق الذي تبنيه. سنتناول في هذه الفقرة كيفية إضافة هذه الميزة إلى مكون المهام.

رغم أنَّ هذه الميزة بسيطة، وتنحصر وظيفتها في إخفاء (أو إظهار) المهام المنجزة، إلا أنَّها ستعطيك فكرة جيدة عن كيفية عمل مثل هذه المزايا.

سنكمل على الشيفرة الأخيرة لتطبيق المهام. ستكون ميزة التصفية على شكل صندوق اختيار Checkbox موجود أعلى قائمة المهام. عندما يختاره المستخدم فإنَّ المهام المنجزة ستختفي، والعكس صحيح.

سنبدأ بإضافة عنصر الاختيار هذا إلى الملف index.html ضمن القسم الخاص بـ قالب المكون. أضف الشيفرة التالية بعد الوسم `<div class='tasks-container'>` مباشرة:

```

<input id="hide_cmp_tasks" type="checkbox" v-
model="hide_completed_tasks"/>
<label for="hide_cmp_tasks">Hide completed tasks</label>

```

لاحظ معي أنَّ هذا العنصر مرتبط بحقل اسمه `hide_competed_tasks` سنعرفه بعد قليل ضمن مكون المهام.

افتح الآن الملف `app.js` وأضف القسمين `data` و `computed` للمكوّن `tasks`. سيحتوي قسم `data` على تعريف الحقل `hide_completed_tasks` المرتبط بعنصر صندوق الاختيار الذي أضفناه قبل قليل، وسيكون من النوع المنطقي `Boolean`، في حين أنّ القسم `computed` فسيحتوي على الخاصية المحسوبة `filtered_tasks`. سيصبح المكوّن `tasks` على الشكل التالي:

```
Vue.component('tasks', {
  template: '#tasks-template',
  props: {
    name: String,
    tasks_list: Array,
  },
  data() {
    return {
      hide_completed_tasks: false
    }
  },
  computed: {
    filtered_tasks() {
      return this.hide_completed_tasks ?
      this.tasks_list.filter(t => !t.done) : this.tasks_list;
    }
  }
})
```

إذا تأملت معي محتويات الخاصية المحسوبة `filtered_tasks` فستجدها عبارة عن شيفرة برمجية بسيطة لتصفية المهام المنجزة من خلال اختبار قيمة الخاصية `done`. أي أنّ عملية التصفية ستحدث في هذا المكان تحديداً بناءً على كون الحقل `hide_completed_tasks` يحمل القيمة `true` أو `false`.

هذا كلّ شيء! إذا أحببت الآن، انقر بزر الفأرة الأيمن على الملف `index.html` الموجود ضمن نافذة المستكشف `Explorer`، واختر الأمر `Open with Live Server` كما اعتدنا من قبل. ستحصل على شكل شبيه بما يلي:

# Welcome to MyTasks Application

This application is built to explain how to deal with components

☐ Hide completed tasks

Husam - Tasks	
Done	Title
<input checked="" type="checkbox"/>	Write an introduction about VueJS components.
<input type="checkbox"/>	Drink a cup of tea.
<input type="checkbox"/>	Call Jamil.
<input checked="" type="checkbox"/>	Buy a new book.

جرب الآن أن تنقر بشكل متكرر على صندوق الاختيار Hide completed tasks لتجد كيف أنَّ المهام المنجزة تختفي وتظهر وفقًا لذلك.

## 5.7 إضافة ميزة مهمة جديدة لمكون المهام

لنحسن مكون المهام بميزة إضافة مهمة جديدة للمهام الموجودة مسبقًا. سأضع أسفل قائمة المهام عنصر إدخال نصي مع زر للإضافة. و سأضيف أيضًا بعض اللمسات باستخدام CSS لكي أحصل على مظهر مقبول لهما.

افتح الملف tasks.css وأضف أصناف CSS التالية له:

```
.add-task-container {
  position: relative;
  margin-top: 10px;
}

.add-task-container div{
  position: absolute;
  top: 0;
  right: 60px;
  left: 45px;
}

.add-task-container div input{
  width: 100%;
```

```

}

.add-task-container input{
  position: absolute;
  width:50px;
  top: 0;
  right: 0;
}

```

انتقل الآن إلى الملف index.html وأضف الشيفرة التالية بعد نهاية وسم الإغلاق للجدول مباشرةً (ضمن

القالب الخاص بمكوّن المهام):

```

<div class="add-task-container">
  <span>Task: </span>
  <div>
    <input type="text" v-model="new_task_text"/>
  </div>
  <input type="button" value="Add" v-on:click="add_new_task"/>
</div>

```

حان الآن دور الشيفرة البرمجية ضمن الملف app.js. افتح هذا الملف، واحرص على أن تكون محتوياته

مطابقة لما يلي:

```

Vue.component('tasks', {
  template: '#tasks-template',
  props: {
    name: String,
    tasks_list: Array,
  },
  data() {
    return {
      hide_completed_tasks: false,
      new_task_text : ""
    }
  },
  computed: {
    filtered_tasks() {
      return this.hide_completed_tasks ?
      this.tasks_list.filter(t => !t.done) : this.tasks_list;
    }
  },
  methods: {
    add_new_task(event) {
      this.tasks_list.push({ title: this.new_task_text, done:
      false });
      this.new_task_text = "";
    }
  }
})

```

```
new Vue({
  el: '#app'
})
```

الجديد هنا هو أنني أضفت حقلاً جديداً إلى القسم `data` وأسميته `new_task_text` سيرتبط بمربع النص الذي سندخل من خلاله عنوان المهمة. كما قد أضفت القسم `methods` وعرفت ضمنه التابع الجديد `add_new_task`، وهذا التابع الذي سيُستدعى عند نقر زر إضافة المهمة.

الشفرة البرمجية الموجودة في هذا التابع بسيطة، فهي تعمل على إضافة المهمة الجديدة إلى مصفوفة المهام `tasks_list` ثم تفرغ عنوان المهمة من جديد لاستقبال المهمة التالية.

جرب كتابة المهمة التالية: `My new task!` ضمن مربع النص، ثم انقر الزر `Add`. ستحصل على شكل

شبيه بما يلي:

## Welcome to MyTasks Application

This application is built to explain how to deal with components

☐ Hide completed tasks

Husam - Tasks	
Done	Title
<input checked="" type="checkbox"/>	Write an introduction about VueJS components.
<input type="checkbox"/>	Drink a cup of tea.
<input type="checkbox"/>	Call Jamil.
<input checked="" type="checkbox"/>	Buy a new book.
<input type="checkbox"/>	My new task!

Task:



## 5.8 ختام الفصل

بدأنا في هذا الفصل بالولوج في عالم المكونات، إذ يُعتبر ما تناولناه في هذا الفصل مقدّمة بسيطة إلا أنها مهمة في التعامل مع المكونات حيث تعلمنا كيف نبني هيكلًا لتطبيق بسيط على حاسوبنا الشخصي، وكيفية إضافة خادوم مبسّط لمحاكاة عمل التطبيق بشكل فعلي، كما بنينا مكونًا مفيدًا، وتعلّمنا أساسيات فصل شيفرة vue.js عن باقي التطبيق. هنالك المزيد لتعلّمه حول المكونات وحول vue.js بصورة عامة في الفصول القادمة إن شاء الله.

## 5.9 تمارين داعمة

أنشئ مكونًا باسم `vu-countdowntimer` وظيفته العد التنازلي بمقدار ثانية واحدة كل مرّة، ابتداءً من قيمة محددة يمكن تمريرها للمكون، وحتى الصفر.

## 6. التعمق في مكونات Vue.js

سنتعلم في هذا الفصل:

- بناء تطبيق نموذجي (مشروبات حسوب).
- إضافة وسائل تنقيح متطورة لتطبيقات Vue.js.
- المكونات المتداخلة.
- تسجيل المكونات محليًا وتسجيلها على المستوى العام.
- تحديد المكون الذي اختاره المستخدم.
- التخاطب بين المكونات باستخدام أحداث مخصصة.

سنتعلم في هذا الفصل المزيد عن المكونات، حيث سنتعرف على المزيد من المزايا المتعلقة بها، والتي ستساعدنا على بناء مكونات عملية ومفيدة. سنبدأ هذا الفصل ببناء تطبيق نموذجي سيكون الهيكل الأساسي الذي سنستخدمه لتعلم المزايا الجديدة حول المكونات، ثم سنتعرف على كيفية استخدام أدوات تنقيح متطورة مكتوبة خصيصًا لـ Vue.js، ثم نتعرف على المكونات المتداخلة وكيفية استخدامها، ثم سنتعلم كيفية تسجيل المكونات محليًا وعلى المستوى العام في التطبيق، ونختم بتعلم كيفية التخاطب بين المكونات المتداخلة.

## 6.1 بناء تطبيق نموذجي (مشروبات حسوب)

سنبني من أجل هذا الفصل تطبيق نموذجي بسيط لكي نطبق عليه الأفكار التي سنتناولها هنا. سيكون هذا التطبيق عبارة عن واجهة بسيطة لمحل افتراضي بيع مشروبات متنوعة. سيعبر المكوّن في هذه المرة عن مشروب معيّن من قائمة المشروبات المتاحة.

انظر إلى الشكل النهائي المقترح:



كما ترى فإنني أستخدم اللغة العربية في التطبيق هذه المرّة! أنشئ مجلّدًا جديدًا سمّاه hsub-drinks سنستخدمه لوضع ملفات المشروع ضمنه.

يحتوي هذا التطبيق على مكوّن وحيد حاليًا أسميته drink أي مشروب ما. لهذا المكوّن خاصيّة وحيدة اسمها name تعبر عن اسم هذا المشروب. انظر إلى شيفرة Vue.js التي سأضعها ضمن الملف app.js الذي سيكون موجودًا ضمن المجلّد hsub-drinks الذي أنشأته تّوًّا:

```
Vue.component('drink', {
  template: '#drink-template',
  props: {
    name: {
      type: String
    }
  }
});

new Vue({
  el: '#app',
```

```

    data: {
      drinks: ["شاي", "قهوة", "شاي أخضر", "زهورات", "با بونج"]
    }
  })

```

لاحظ كم هو بسيط هذا التطبيق: مكون عادي معرّف في الأعلى، وكائن Vue.js بسيط في الأسفل يحتوي على البيانات التي نريد عرضها على الشاشة. هذه البيانات موجودة ضمن المصفوفة drinks كما هو واضح.

لننتقل الآن إلى شيفرة HTML التي سأضعها ضمن ملف سائمه index.html وسيكون موجودًا أيضًا

ضمن المجلد hsub-drinks:

```

<html lang="ar">
<body>
  <div class="header">
    <span id="logo">مشروبات حسب</span>
  </div>

  <div id="app" class="container">
    <div class="content">
      <h1 class="title">المشروبات المتوفرة</h1>

      <div class="drinks">
        <drink v-for="drink in drinks" v-
bind:name="drink"></drink>
      </div>
    </div>
  </div>

  <script type="text/x-template" id="drink-template">
    <div class="drink">
      <div class="description">
        <span class="title">
          {{name}}
        </span>
      </div>
    </div>
  </script>

  <script src="https://unpkg.com/vue@2.6.11/dist/vue.js"></script>
  <script src="app.js"></script>
  <link rel="stylesheet" href="app.css" />

</body>
</html>

```

كما مرّ معنا فيما سبق نستخدم حلقة `v-for` في المرور على عناصر المصفوفة `drinks` وبالتالي توليد عنصر المكوّن `drink` كما هو واضح من الشيفرة السابقة. لاحظ أيضًا أنّ العناصر المولّدة والتي تمثّل المشروبات المتوفرة ستكون موجودة ضمن عنصر `div` يحمل الصنف `drinks`. أرجو أن لا يختلط عليك الأمر بين اسم المكوّن `drink` وبين صنف التنسيق `drink` لأنّ كليهما يحملان نفس الاسم، وهذا أمر جائز تمامًا.

بقيت أخيرًا تنسيقات CSS الذي سأضعها ضمن الملف `app.css` ضعه أيضًا ضمن

المجلد `hsoub-drinks`:

```
@import url(//fonts.googleapis.com/earlyaccess/notonaskharabic.css);
body {
  height: 100vh;
  -webkit-font-smoothing: auto;
  -moz-osx-font-smoothing: auto;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  font-family: 'Noto Naskh Arabic', serif;
  background-color: #ccdcdc;
  background-repeat: no-repeat;
  background-position: 100% 100%
}

span#logo {
  font-weight: 700;
  color: #eee;
  font-size: larger;
  letter-spacing: .05em;
  padding-left: .5rem;
  padding-right: .5rem;
  padding-bottom: 1rem;
  float: right;
  padding-top: 6px;
  margin-right: 20px;
}

.header{
  background-color:slategray;
  width: 80% ;
  height: 50px;
  margin-left: auto;
  margin-right: auto;
}

h1.title {
  text-align: center;
  font-size: 1.875rem;
```

```

    font-weight: 500;
    color: #2d3336
  }

  h2.subtitle {
    margin: 8px auto;
    font-size: x-large;
    text-align: center;
    line-height: 1.5;
    max-width: 500px;
    color: #5c6162
  }

  .content {
    margin-left: auto;
    margin-right: auto;
    padding-top: 1.5rem;
    padding-bottom: 1.5rem;
    width: 620px
  }

  .drinks {
    padding: 0 40px;
    margin-bottom: 40px
  }

  .drinks .drink {
    background-color: #fff;
    -webkit-box-shadow: 0 2px 4px 0 rgba(0, 0, 0, .1);
    box-shadow: 0 2px 4px 0 rgba(0, 0, 0, .1);
    margin-top: 1rem;
    margin-bottom: 1rem;
    border-radius: .25rem;
    -webkit-box-pack: justify;
    -ms-flex-pack: justify;
    justify-content: space-between;
    cursor: pointer;
    position: relative;
    -webkit-transition: all .3s ease;
    transition: all .3s ease
  }

  .drinks .drink, .drinks .drink>.weight {
    display: -webkit-box;
    display: -ms-flexbox;
    display: flex
  }

  .drinks .drink>.description {
    width: 100%;
    padding: 1rem;
  }

```

```
.drinks .drink>.description .title {
  color: #3d4852;
  display: block;
  font-weight: 700;
  margin-bottom: .25rem;
  float: right;
}

.drinks .drink>.description .description {
  font-size: .875rem;
  font-weight: 500;
  color: #8795a1;
  line-height: 1.5
}

.drinks .drink>.price {
  width: 20%;
  color: #09848d;
  display: -webkit-box;
  display: -ms-flexbox;
  display: flex;
  padding-top: 1.5rem;
  font-family: Crimson Text, serif;
  font-weight: 600
}

.drinks .drink>.price .dollar-sign {
  font-size: 24px;
  font-weight: 700
}

.drinks .drink>.price .number {
  font-size: 72px;
  line-height: .5
}

.drinks .active-drink, .drinks .drink:hover {
  -webkit-box-shadow: 0 15px 30px 0 rgba(0, 0, 0, .11), 0 5px 15px 0
  rgba(0, 0, 0, .08);
  box-shadow: 0 15px 30px 0 rgba(0, 0, 0, .11), 0 5px 15px 0 rgba(0,
  0, 0, .08)
}

.drinks .active-drink:after, .drinks .drink:hover:after {
  border-width: 2px;
  border-color: #7dacaf;
  border-radius: .25rem;
  content: "";
  position: absolute;
  display: block;
  top: 0;
```

```

    bottom: 0;
    left: 0;
    right: 0
  }

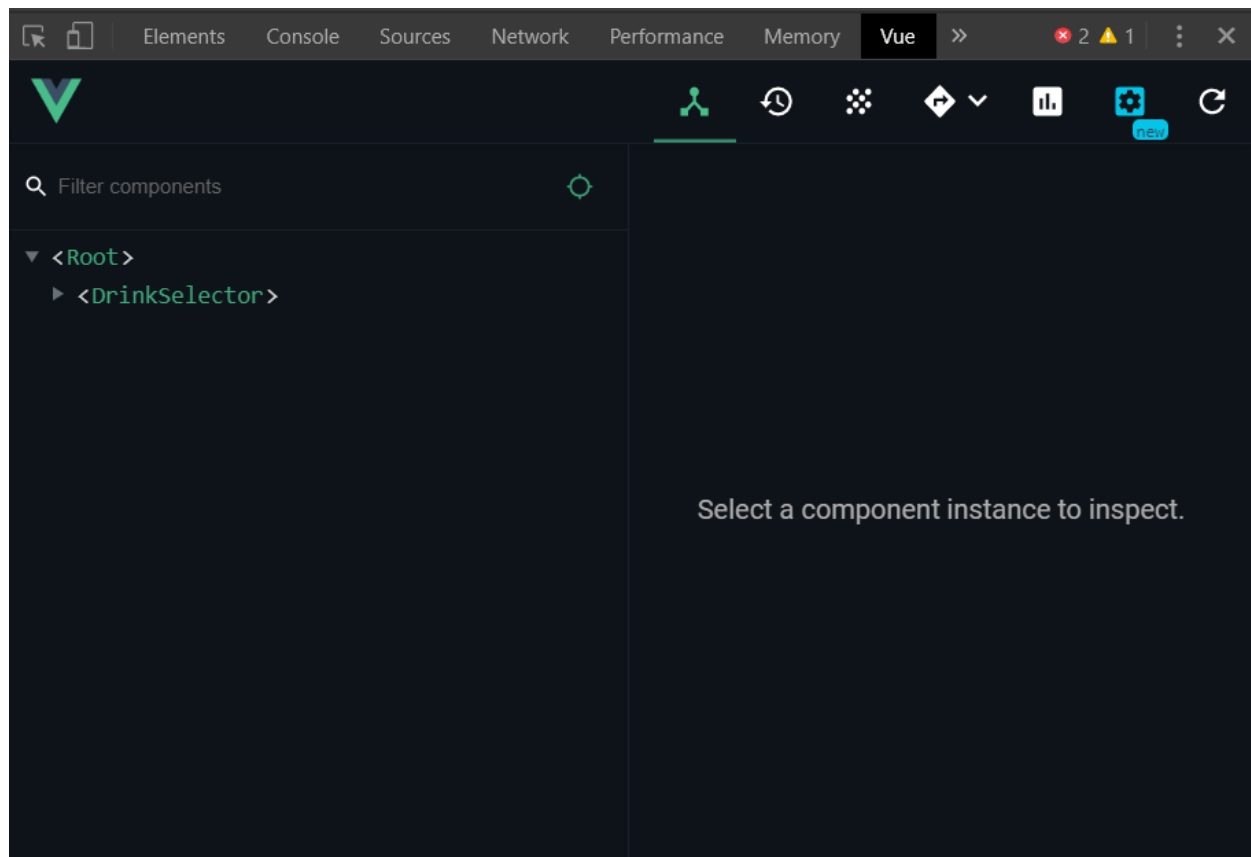
```

أصبح التطبيق النموذجي جاهزًا. نستطيع الآن المتابعة مع موضوعات هذا الفصل.

## 6.2 إضافة وسائل تنقيح متطورة لتطبيقات Vue.js

يوفر مطورو Vue.js أدوات تنقيح متطورة مخصصة لتطبيقات Vue.js بحيث تسهل حياة المبرمج إلى حد كبير. تظهر هذه الأدوات ضمن أدوات المطور التي يمكن الوصول إليها بضغط المفتاح F12. يمكنك زيارة [هذه الصفحة](#) للاطلاع على كيفية تثبيت هذه الأدوات.

بعد أن تثبت الأدوات السابقة، يمكنك الوصول إليها بضغط المفتاح F12 كما أشرنا، ثم تبحث عن لسان التبويب Vue.js (قد تحتاج إلى نقر زر عرض المزيد في حال لم تجد لسان التبويب هذا). انظر الشكل التالي لترى كيف تبدو.





## 6.3 المكونات المتداخلة

نحتاج في بعض الأحيان إلى جعل المكونات متداخلة فيما بينها، كأن يكون هناك مكون أساسي (مكون أب) يستخدم مكونات أصغر (مكونات أبناء) ضمنه. سنوظف هذا المفهوم ضمن التطبيق النموذجي السابق. سأضيف مكونًا جديدًا ضمن الملف `app.js` وسأسمّيه `drink-selector`. سيلعب هذا المكون دور المكون الأساسي الذي يحوي المكون `drink` الذي سيكون ضمنه بشكل متداخل. انظر إلى محتويات الملف `app.js` بعد إضافة المكون الجديد إليه:

```
Vue.component('drink', {
  template: '#drink-template',
  props: {
    name: {
      type: String
    }
  }
})

Vue.component('drink-selector', {
  template: '#drink-selector-template',
  data() {
    return {
      drinks: ["شاي", "قهوة", "شاي أخضر", "زهورات", "با بونج"]
    }
  }
})

new Vue({
  el: '#app'
})
```

لاحظ معي كيف نقلت البيانات الخاصة بالتطبيق من كائن `Vue.js` إلى المكون `drink-selector`. لهذا الأمر فائدة سنراها لاحقًا إن شاء الله.

بالنسبة لشفيرة HTML فالأمر بسيط أيضًا. سننشئ القالب `drink-selector-template` المعتبر عن مكوننا الجديد. وسننقل إليه الشفيرة الخاصة بتوليد قائمة المشروبات. انظر محتوى القالب:

```
<script type="text/x-template" id="drink-selector-template">
  <div class="drinks">
    <drink v-for="drink in drinks" v-
bind:name="drink"></drink>
  </div>
</script>
```

بدلاً من الشيفرة التي نقلناها تَوّأ يمكنك استخدام العنصر `<drink-selector>`. انظر كيف ستصبح الشيفرة بعد التعديل الأخيرة على الملف `index.html`:

```
<html lang="ar">

<body>
  <div class="header">
    <span id="logo">مشروبات حسوب</span>
  </div>

  <div id="app" class="container">
    <div class="content">
      <h1 class="title">المشروبات المتوفرة</h1>
      <drink-selector></drink-selector>

    </div>
  </div>

  <script type="text/x-template" id="drink-selector-template">
    <div class="drinks">
      <drink v-for="drink in drinks" :name="drink"></drink>
    </div>
  </script>

  <script type="text/x-template" id="drink-template">
    <div class="drink">
      <div class="description">
        <span class="title">
          {{name}}
        </span>
      </div>
    </div>
  </script>

  <script src="https://unpkg.com/vue@2.6.11/dist/vue.js"></script>
  <script src="app.js"></script>
  <link rel="stylesheet" href="app.css" />

</body>

</html>
```

إذا استطعنا استخدام مكوّن ضمن مكوّن آخر بشكل متداخل. يمكنك الآن أن تنشئ عناصر `drink-selector` جديدة بنسخ ولصق السطر التالي بشكل متكرر:

```
<drink-selector></drink-selector>
```

## 6.4 تسجيل المكونات محليًا وتسجيلها على المستوى العام

يمكن تسجيل المكونات ضمن تطبيق Vue.js بأسلوبين مختلفين. الأسلوب الأول هو الأسلوب العام، وهو الأسلوب الذي استخدمناه حتى هذه اللحظة باستخدام التابع `Vue.component`. والأسلوب الآخر هو الأسلوب المحلي والذي سنتعرف عليه في هذه الفقرة.

في الحقيقة لا يُعتبر تسجيل المكونات على المستوى العام أمرًا جيّدًا، لأنّه عندما ستكبر تطبيقاتك وتبدأ باستخدام أساليب متقدمة في بناء التطبيقات (كما سيمر معنا في الفصول اللاحقة)، سيتم بناء المكونات التي سجلتها على المستوى العام وذلك في التطبيق النهائي، حتى ولو لم تستخدمها في ذلك التطبيق. وهذا يعني زيادة في حجم شيفرة JavaScript التي على المستخدمين تحميلها من الإنترنت دون فائدة. هذا فضلًا عن مشاكل من الناحية التصميمية للتطبيق. ففي تطبيقنا الأخير مثلاً، لن نستخدم المكون `drink` خارج المكون `drink-selector`، وبالتالي لا حاجة لتسجيل المكون `drink` على المستوى العام.

لحل هذه المشكلة، وبالتالي تسجيل المكونات بشكل محلي إذا اقتضى الأمر ذلك، فيمكننا بكل بساطة تعريف المكون على شكل كائن JavaScript وإسناده إلى متغير عادي بدون استخدام التابع `Vue.Component`. ثم تسجيله في المكان الذي سنستخدمه فيه فقط. دعنا نطبق هذه الطريقة على المكون `drink` الذي سيصبح تعريفه على النحو التالي:

```
let drink_component = {
  template: '#drink-template',
  props: {
    name: {
      type: String
    }
  }
}
```

بما أنّ هذا المكون سيستخدم ضمن المكون `drink-selector` لذلك سننشئ قسمًا جديدًا ضمن المكون `drink-selector` اسمه `components` والذي يسمح بتسجيل أي مكونات داخلية سيستخدمها هذا المكون. انظر إلى تعريف المكون `drink-selector` بعد التعديل:

```
Vue.component('drink-selector', {
  template: '#drink-selector-template',
  components: {
    drink: drink_component
  },
  data() {
    return {
```

```

      drinks: ["شاي", "قهوة", "شاي أخضر", "زهورات", "با بونج"]
    }
  }
})

```

وبذلك نكون قد سجلنا المكوّن `drink` محليًا ضمن المكوّن `drink-selector` ولا يمكن بعد ذلك استخدام المكون `drink` في أي مكان آخر غير المكون `drink-selector`.

كما يمكن بطبيعة الحال فعل الأمر ذاته مع المكوّن `drink-selector` أي تسجيله محليًا بدون استخدام التابع `Vue.component` في حال أردنا استخدام هذا المكوّن فقط ضمن صفحة محدّدة ضمن تطبيق الويب. سنجر الآن تعديلًا مماثلًا على المكوّن `drink-selector`:

```

let drink_selector_component = {
  template: '#drink-selector-template',
  components: {
    drink: drink_component
  },
  data() {
    return {
      drinks: ["شاي", "قهوة", "شاي أخضر", "زهورات", "با بونج"]
    }
  }
}

```

الآن، أين تعتقد أنه يجب تسجيل المكوّن `drink-selector`؟

الجواب بكل بساطة، وبما أنه لا يوجد مكوّن رئيسي يمكن أن يُعرّف المكوّن `drink-selector` ضمنه، فسنسجله ضمن كائن `Vue.js` الخاص بالتطبيق باستخدام القسم `components` أيضًا:

```

new Vue({
  el: '#app',
  components: {
    'drink-selector': drink_selector_component
  }
})

```

لاحظ معي أنني قد عرفت اسم المكوّن هذه المرة على شكل نص: `drink-selector` بشكل مختلف عن تعريف المكوّن `drink`. السبب في ذلك أنني أرغب بالاستمرار باستخدام الرمز - ضمن اسم المكون `drink-selector` وهذا جائز تمامًا بالطبع. إذا أردت الاطلاع على الشكل النهائي للملف `app.js` بعد التعديلات الأخيرة، انظر إلى الشيفرة التالية:

```

let drink_component = {
  template: '#drink-template',
  props: {
    name: {
      type: String
    }
  }
}

let drink_selector_component = {
  template: '#drink-selector-template',
  components: {
    drink: drink_component
  },
  data() {
    return {
      drinks: ["شاي", "قهوة", "شاي أخضر", "زهورات", "با بونج"]
    }
  }
}

new Vue({
  el: '#app',
  components: {
    'drink-selector': drink_selector_component
  }
})

```

## 6.5 تحديد المكون الذي اختاره المستخدم

لنمضي قدمًا في تطبيق المشروبات، حيث سنعمل الآن على إضافة ميزة تحديد المشروب الذي اختاره المستخدم عن طريق النقر عليه بالفأرة. تحتاج هذه الميزة إلى بعض المتطلبات البسيطة، حيث سُنكسب المكون `drink` حقلًا جديدًا اسمه `is_selected` سيُعرّف ضمن القسم `data` كما نعلم، لتحديد فيما إذا كان المستخدم قد اختار هذا المشروب أم لا، بالإضافة إلى تزويده بتابع جديد يسمح باختيار المشروب ولنسمّه `select` والذي سيحتوي على تعليمة برمجية واحدة تجعل قيمة الحقل `is_selected` مساوية للقيمة `true`، أي تعبر عن عملية الاختيار. انظر كيف سيبدو شكل المكون `drink` بعد إضافة التعديلات السابقين عليه:

```

let drink_component = {
  template: '#drink-template',
  props: {
    name: {
      type: String
    }
  }
}

```

```

    },
    data() {
      return {
        is_selected: false
      }
    },
    methods: {
      select() {
        this.is_selected = true;
      }
    }
  }
}

```

لنجر الآن التعديلات اللازمة ضمن الملف index.html حيث سنضيف الموجه v-on:click ضمن قالب

المكوّن drink للاستجابة لحدث النقر. انظر التعديلات التالية على القالب drink-template:

```

<script type="text/x-template" id="drink-template">
  <div v-on:click="select" class="drink">
    <div class="description">
      <span class="title">
        {{name}}
      </span>
    </div>
  </div>
</script>

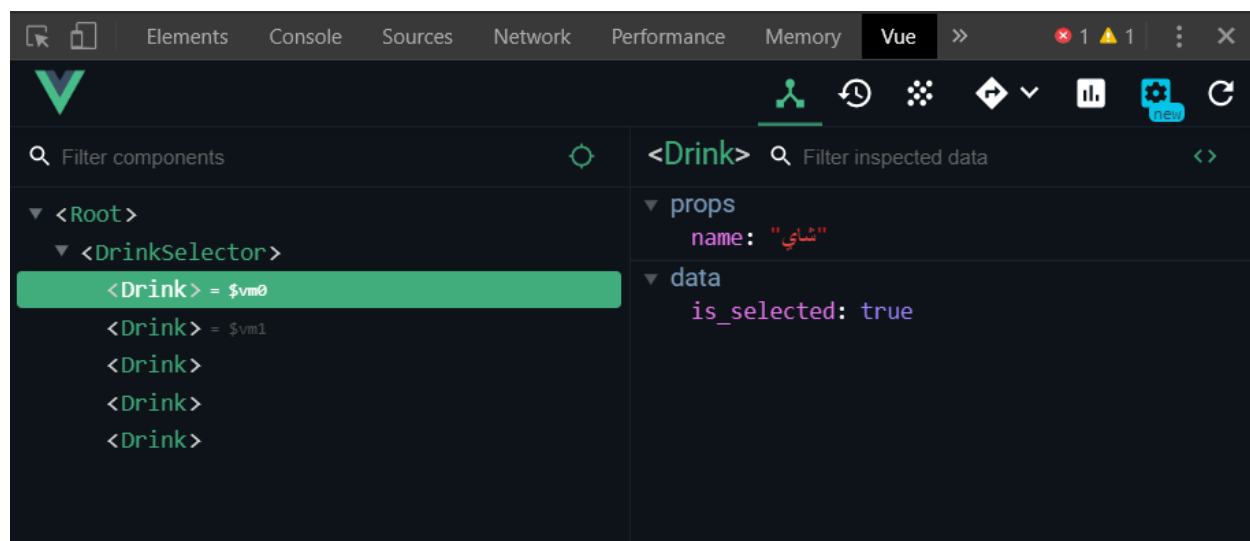
```

عد إلى المتصفح لمعاينة التغييرات (تذكر أننا نستعرض الملف index.html ضمن

الخادوم Live Server). جرب أن تنقر على المشروبين: "شاي" و "شاي أخضر"، ثم انتقل إلى نافذة أدوات

المطور بضغط F12، ومن هناك انتقل إلى لسان التبويب Vue. انشر العقد في حال احتجت ذلك، ستحصل على

شكل شبيه بما يلي بالنسبة للمشروب الأول "شاي":



جرب أن تعالين حالة المشروب التالي "قهوة" ستجد أن قيمة الحقل `is_selected` له تساوي `false` لأننا لم ننقر على القهوة. جرب الآن أن تعالين حالة المشروب التالي "شاي أخضر"، ستلاحظ أن قيمة `is_selected` في هذه الحالة هي `true`. أي أن التطبيق حاليًا يعمل كما هو متوقع منه حتى الآن. بقي أن نضيف بعض التأثيرات البصرية البسيطة للإشارة إلى أن المشروب قد اختير من قبل المستخدم. سأستخدم الموجّه `v-bind:class` لإضافة تنسيق CSS وذلك ضمن القالب `drink-template` على النحو التالي:

```
<script type="text/x-template" id="drink-template">
  <div v-on:click="select" class="drink" v-bind:class="{ 'active-
drink': is_selected }">
    <div class="description">
      <span class="title">
        {{name}}
      </span>
    </div>
  </div>
</script>
```

سيُطبق تنسيق CSS المسمى 'active-drink' فقط عندما تكون قيمة الحقل `is_selected` تساوي `true` أي عندما يختار المستخدم المشروب الحالي. ولا حاجة بطبيعة الحال لاستخدام علامة الاقتباس المفردة في حال لم يحتوي اسم الصنف على رمز مثل `-`. بقي أن نضيف تنسيق CSS نفسه إلى ملف التنسيقات `app.css`. أضف الصنفين التاليين إلى ذلك الملف:

```
.drinks .active-drink, .drinks .drink:hover {
  background-color: lightgray;
  -webkit-box-shadow: 0 15px 30px 0 rgba(0, 0, 0, .11), 0 5px 15px 0
  rgba(0, 0, 0, .08);
  box-shadow: 0 15px 30px 0 rgba(0, 0, 0, .11), 0 5px 15px 0 rgba(0,
  0, 0, .08)
}

.drinks .active-drink:after, .drinks .drink:hover:after {
  border-width: 2px;
  border-color: #7daca;
  border-radius: .25rem;
  content: "";
  position: absolute;
  display: block;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0
}
```

عد إلى المتصفح واختر المشروب الأول والثالث مرة أخرى لتحصل على شكل شبيه بما يلي:

المشروبات المتوفرة

شاي

قهوة

شاي أخضر

زهورات

بابونج

لقد تمّت المهمة بنجاح! ولكن، ماذا لو أردنا اختيار مشروب واحد فقط من القائمة؟ سنحل هذه المشكلة في الفقرة التالية حيث سنتعلّم كيفية التخاطب بين المكونات المختلفة باستخدام أحداث مخصصة.

## 6.6 التخاطب بين المكونات باستخدام أحداث مخصصة

برزت الحاجة في الفقرة السابقة إلى اختيار مشروب واحد فقط من قائمة المشروبات المتاحة. أي أننا بحاجة إلى آلية معينة تسمح باختيار مشروب واحد فقط، بحيث تعمل هذه الآلية على إلغاء أي اختيار سابق لمشروب ما (في حال وجوده) ضمن القائمة، والإبقاء على المشروب الذي اختاره المستخدم أخيراً.

يمكن تحقيق هذا الأمر بالتخاطب بين المكوّن الابن `drink` والمكوّن الأب `drink-selector` حيث سنستخدم المكوّن الأب للمساعدة في هذا الأمر.

يمكن التخاطب بين المكونات باستخدام الأحداث المخصصة التي سنتحدث عنها بعد قليل، حيث سنبذل المكوّن الأب `drink-selector` بأن مشروباً ما قد اختاره المستخدم، فيعمل عندئذ المكوّن الأب على إلغاء اختيار أي مشروب سابق يمكن أن يكون قد اختير من قبل، باستثناء المشروب الحالي.

قبل المتابعة، توقف عن القراءة قليلاً، حضر كوباً من الشاي (أو القهوة)، ثم عد إلى هنا من جديد، لأنّ الشرح التالي يحتاج إلى المزيد من التركيز.



سنبدأ من الملف `app.js` حيث سنجري بعض التعديلات على المكونين `drink` و `drink-selector`. بالنسبة للمكون `drink` سأجري التعديلات التالية:

- استخدمنا في المثال في الفقرة السابقة الحقل `is_selected` للإشارة إلى كون المشروب الحالي قد اختير أم لا. سأحول هذا الحقل إلى خاصية محسوبة (Computed Property) بالإضافة إلى أنني سأحذف قسم `data` بشكل كامل، وسنرى سبب ذلك بالإضافة بعد قليل.
- كما سأستخدم لأول مرة التابع المضمن `$emit` ضمن التابع `select` (بدلاً من التعليمة `this.is_selected = true`) لكي تُصدر الحدث `drink_selected_event` وذلك لكي نبّغ المكون الأب بأن المشروب الحالي قد اختير، وذلك بأن أمرر اسم هذا المشروب كوسيط ثانٍ للتابع `$emit`.
- سأضيف أيضاً خاصية جديدة اسمها `selectedDrink` إلى المكون `drink` ضمن القسم `props` وذلك لكي أسمح للمكون الأب فيما بعد، بتمرير اسم المشروب (مكون `drink`) الذي اختاره المستخدم حالياً إلى هذا المكون.

تأمل الشيفرة البرمجية الجديدة الخاصة بالمكون `drink`:

```
let drink_component = {
  template: '#drink-template',
  props: {
    name: {
      type: String
    },
    selectedDrink: {
      type: String
    }
  },
  computed: {
    is_selected() {
      return this.selectedDrink === this.name;
    }
  },
  methods: {
    select() {
      this.$emit('drink_selected_event', this.name)
    }
  }
}
```

أما بالنسبة للمكوّن drink-selector فسأجري عليه أيضًا التعديلات التالية:

- وضعت ضمنه القسم methods لكي أستطيع وضع التابع drink\_selected\_handler والذي سيكون معالجًا للحدث drink\_selected\_event الذي سأصدره (باستخدام التابع المضمن \$emit) ضمن التابع select في المكوّن drink كما رأينا قبل قليل.
- أضفت حقلاً جديداً أسميته current\_drink ضمن القسم data وهو الذي سيحتفظ باسم المكوّن (المشروب) الذي اختاره المستخدم توّاً.

إليك الشيفرة البرمجية الجديدة الخاصة بالمكون drink-selector:

```
let drink_selector_component = {
  template: '#drink-selector-template',
  components: {
    drink: drink_component
  },
  data() {
    return {
      drinks: ["شاي", "قهوة", "شاي أخضر", "زهورات", "با بونج"],
      current_drink: null
    }
  },
  methods: {
    drink_selected_handler(drink_name) {
      this.current_drink = drink_name;
    }
  }
}
```

إليك الآن التعديلات التي حدثت ضمن قالب المكوّن drink-selector:

```
<script type="text/x-template" id="drink-selector-template">
  <div class="drinks">
    <drink v-for="drink in drinks" v-bind:name="drink"
      v-on:drink_selected_event="drink_selected_handler"
      v-bind:selectedDrink="current_drink"></drink>
  </div>
</script>
```

أريد هنا التركيز على أمرين هما في صلب الموضوع:

- الأمر الأول هو استخدامي للموجّه v-on:drink\_selected\_event. هذا الموجّه ليس مبيّنًا بالأصل. إنما هو موجّه جديد مُحدث بسبب استخدامي للتابع \$emit وتمريري

للقيمة 'drinkselectedevent' له. إذاً يمكن إنشاء موجّهات مخصصة باستخدام التابع المبيّت `$.emit`.

- الأمر الثاني هو الموجّه `v-bind:selectedDrink="current_drink"` الذي يربط قيمة الحقل `current_drink` من الموجّه الأب، مع الخاصية `selectedDrink` من الموجّه الابن.

فالذي يحدث بكل بساطة، هو أنّه عندما يختار المستخدم مشروباً ما من القائمة، فسيستدعى في البداية التابع `select` من المكون الابن. ضمن هذا التابع لتنفيذ التعليمات البرمجية:

```
this.$emit('drink_selected_event', this.name)
```

التي تعمل على استدعاء حدث مخصص للتخاطب مع المكون الأب، حيث يقول المكون الابن للمكون الأب: "انظر لقد تمّ اختياري!"، علماً أنّ اسم المكون الابن (اسم المشروب) سيمرّر ضمن الوسيط الثاني من التابع `$emit` كما أوضحنا قبل قليل.

بسبب وجود الموجّه `v-on:drink_selected_event="drink_selected_handler"` ضمن قالب المكون الأب، سيُلْتَقَط هذا الحدث، وسيستدعى التابع `drink_selected_handler` من المكون الأب، حيث تعمل تعليمة بسيطة ضمنه على إسناد اسم المشروب (المكون الابن) الذي اختاره المستخدم ضمن الحقل `current_drink` وبهذه الطريقة يعرف المكون الأب من هو المشروب الحالي الذي اختاره المستخدم.

ستؤدي هذه المعرفة، وبسبب طبيعة Vue.js إلى تحديث قالب المكون بكامله، وبالتالي ستنفّذ حلقة `v-for` مرة أخرى، ولكن في هذه الحالة سيمرّر اسم المكون (المشروب) الذي اختاره المستخدم إلى جميع المشروبات الموجودة ضمن المكون الأب عن طريق الخاصية `selectedDrink` وبالتالي سيعرف كل مكون ابن من المكون الابن "المحظوظ" الذي اختاره المستخدم حالياً. وبسبب وجود الخاصية المحسوبة `is_selected` في المكون الابن، ستنفّذ التعليمات البرمجية الموجودة ضمنها:

```
return this.selectedDrink === this.name;
```

هذه التعليمة بسيطة للغاية، وهي تُرجع قيمة من النوع المنطقي `true` إذا كان اسم المكون الحالي يطابق اسم المكون "المحظوظ" الذي اختاره المستخدم. أو تُرجع القيمة `false` إذا لم تتحقق هذه المطابقة.

زبدة القول، فإنّ القيمة التي سترجعها الخاصية المحسوبة `is_selected` هي التي ستُسند إلى الموجّه زبدة القول، فإنّ القيمة التي سترجعها الخاصية المحسوبة `is_selected` هي التي ستُسند إلى الموجّه اسم المكون "المحظوظ" الذي اختاره المستخدم. أو تُرجع القيمة `false` إذا لم تتحقق هذه المطابقة.

زبدة القول، فإنّ القيمة التي سترجعها الخاصية المحسوبة `is_selected` هي التي ستُسند إلى الموجّه اسم المكون "المحظوظ" الذي اختاره المستخدم. أو تُرجع القيمة `false` إذا لم تتحقق هذه المطابقة.

تم اختياره بتطبيق تنسيق CSS المناسب أو يظهر بشكل عادي.

نستنتج مما سبق، أنه يمكن للمكوّن الأب أن يرسل رسائل إلى المكوّن الابن الموجود ضمنه، عن طريق الخصائص التي نعرفها ضمن القسم props في المكون الابن، التي تُعتبر مستقبلات للمكون الابن تسمح له بالتقاط الرسائل من خارجه بشكل عام. أما عندما يريد المكون الابن مخاطبة المكون الأب، فيمكن ذلك عن طريق أحداث مخصصة باستخدام التابع `$emit` كما أوضحنا في المثال الأخير.

جرب التطبيق بعد التعديلات الأخيرة. لاحظ كيف أصبح يسمح باختيار مشروب واحد فقط.

## 6.7 ختام الفصل

لقد تناولنا في هذا الفصل الكثير من الموضوعات المهمة حول المكونات، واعتقد أنه أصبح من الضروري الانتقال إلى مستوى أعلى، عن طريق بناء تطبيقات بأساليب احترافية توفرها Vue.js لنا. سنعمل في الفصل القادم إن شاء الله على الاطلاع على كيفية فصل المكونات ضمن ملفات مستقلة تحتوي الواحدة منها على جميع التفاصيل المتعلقة بالمكوّن مما يساهم بتنظيم الشيفرة البرمجية إلى حدّ كبير. بالإضافة إلى الاطلاع على تقنية بناء تطبيقات Vue.js بشكل مختلف عن طريق Vue.js CLI. حتى الفصل القادم أرجو أن تكونوا بأفضل حال!

## 7. إنشاء مشاريع باستخدام Vue CLI

سنتعلم في هذا الفصل:

- لماذا Vue CLI؟
  - إعداد Vue CLI
  - إنشاء مشروع جديد باستخدام Vue CLI
  - نظرة عامة على هيكل المشروع
  - تعديل تطبيق مشروبات حسب الأسلوب الجديد
- سنتعرف في هذا الفصل على Vue CLI ولماذا نحتاج إليه. وسنتعرف أيضًا على كيفية إعداد Vue CLI بالتفصيل، ثم سنعمل على موائمة تطبيق "مشروبات حسب" الذي بنيناه في الفصل السابق لكي يتوافق مع Vue CLI.

## 7.1 لماذا Vue CLI؟

بدأنا أول فصل في الكتاب بكتابة التطبيقات التي نحتاج لها ضمن موقع JSFiddle، ثم طورنا عملنا بأن استخدمنا محرر الشيفرة البرمجية Visual Studio Code بحيث أصبحنا نكتب تطبيقاتنا محليًا باستخدامه، ثم نجرب هذه التطبيقات عن طريق الإضافة Live Server.

السيناريو الأخير جيد في الواقع ولكن قد يكون في بعض الأحيان غير كاف. وخاصةً إذا بدأت بكتابة تطبيقات أكبر وأكثر تعقيدًا باستخدام المكونات التي تكتبها أنت والتي تحتاج إليها في تطبيقاتك.

المشكلة التي ستواجهها هي التداخل بين الشيفرة الخاصة بالمكونات مع الشيفرة الخاصة بالتطبيق، وهذا يحدث على مستوى شيفرة JavaScript و شيفرة HTML وحتى تنسيقات CSS. في حال التطبيقات الكبيرة سيكون هذا التداخل مزعج وأكثر عرضة للأخطاء. لذلك فمن الحكمة أحيانًا الانتقال إلى مستوى أعلى في بناء تطبيقات عملية وواقعية باستخدام أدوات تطوير فعالة ومناسبة.

بغية تحقيق هذا الهدف، سنستخدم في هذا الفصل Vue CLI وهي واجهة موجهة الأوامر الخاص بـ Vue.js. باختصار، هي عبارة عن حزمة برمجية تسمح لنا ببناء الهيكل الأساسي لتطبيق Vue.js تمهيديًا لكتابة الشيفرة، بالإضافة إلى تزويدنا بخادوم تطوير بسيط (مختلف عن الخادوم Live Server) يسمح لنا بمحاكاة الظروف الفعلية التي سيعمل ضمنها هذا التطبيق. توجد عدة قوالب متاحة يستطيع Vue.js توليدها لك، سنستخدم أبسطها في هذا الفصل.

## 7.2 إعداد Vue CLI

نستطيع تثبيت Vue CLI على مختلف أنواع أنظمة التشغيل الأساسية (Windows, Linux, Mac OS) بنفس الأسلوب تقريبًا. سأركز هنا على نظام التشغيل Windows.

### 7.2.1 تثبيت Node.js

رغم أننا سنحتاج إلى Node.js إلا أننا لن نكتب أي شيفرة باستخدامه حاليًا، إنما سنستخدم مدير الحزم npm وهو ضروري لتثبيت Vue CLI، بالإضافة إلى احتوائه على خادوم التطوير الذي تحدثنا عنه قبل قليل، والذي سيستضيف تطبيقاتنا أثناء تطويرها.

سنحتاج إلى Node.js وتأكد أنه مثبت لديك، أو زر الموقع الخاص بـ Node.js ونزل الإصدار الأخير منه (يفضل استعمال الإصدار LTS المستقر طويل الدعم ولكن يمكنك تنزيل أحدث إصدار Current).

بعد تنزيل برنامج التثبيت، شغل هذا البرنامج (ستحتاج إلى صلاحيات مدير النظام)، واتبع خطوات التثبيت مع ترك الخيارات الافتراضية كما هي. ستأخذ عملية التثبيت زمنًا قصيرًا نسبيًا. بعد الانتهاء يمكنك الانتقال إلى الخطوة التالية.

## 7.2.2 تثبيت Vue CLI

شغل موجّه الأوامر في Windows (يمكنك ذلك من خلال ضغط مفتاح الويندوز مع المفتاح R، ثم كتابة الأمر cmd مباشرةً، ونقر زر موافق OK). بعد ذلك اكتب الأمر التالي:

```
npm install -g @vue/cli
```

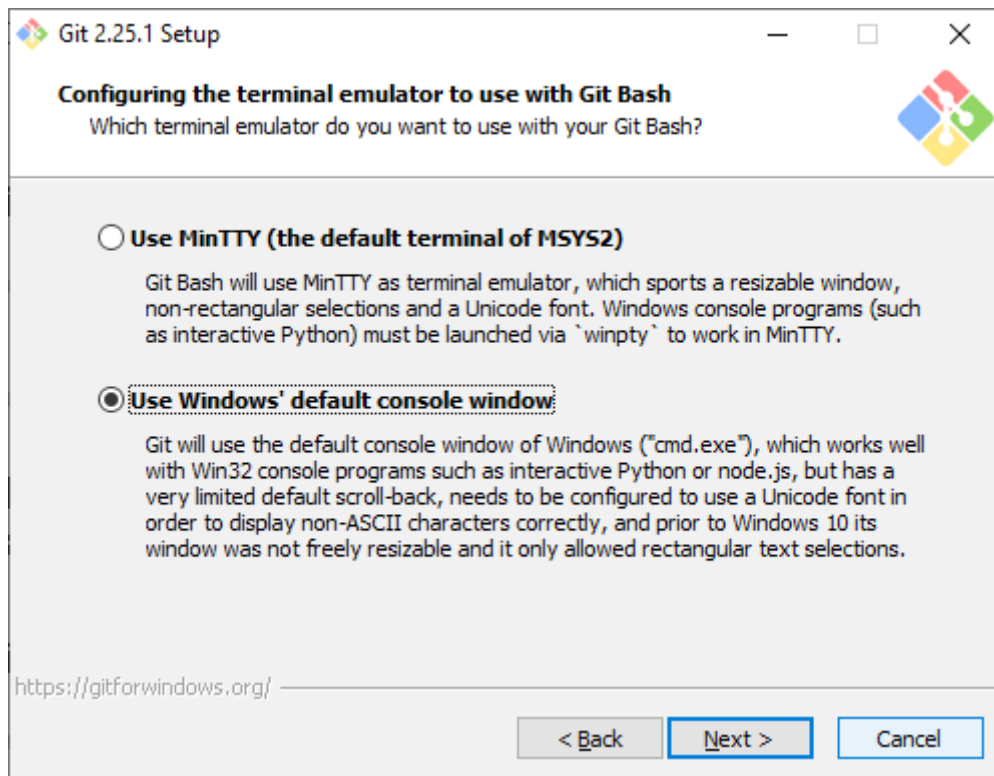
ستبدأ عندها عملية تثبيت Vue CLI والتي ستأخذ أيضًا وقتًا قصيرًا نسبيًا. بعد الانتهاء، سيعود موجّه الأوامر إلى حالته الطبيعية.

عند هذه النقطة أصبح Vue CLI جاهزًا للاستخدام، حيث يمكنك البدء بتنفيذ الأمر vue مباشرةً ضمن موجّه الأوامر لأنه أصبح متاحًا بعد تثبيته باستخدام npm.

قد تحتاج إلى إغلاق نافذة موجّه الأوامر الحالية، وفتح نافذة موجّه أوامر جديدة حتى تستطيع استخدام الأمر vue.

## 7.2.3 تثبيت Git

سنحتاج أيضًا إلى تطبيق إدارة الإصدار Git. يمكنك تحميله من [هذا الرابط](#) مع اختيار نسخة ويندوز. بعد تنزيل الملف، نصب التطبيق (ستحتاج إلى صلاحيات مدير النظام). اترك الخيارات الافتراضية كما هي باستثناء الشاشة التي تطلب منك فيها تحديد الطرفية الافتراضية لـ Git. اختر الخيار الثاني Use Windows' default console window. ثم تابع بنقر الزر Next حتى تصل للنهاية.



بعد الانتهاء من تثبيت Git. افتح نافذة موجه الأوامر (نافذة جديدة) واكتب الأمر التالي لكي نتأكد من أنَّ عملية التثبيت قد تمَّت بشكل صحيح:

```
git --version
```

ستحصل على إصدار النسخة الحالي إذا جرت الأمور بشكل سليم.

## 7.3 إنشاء مشروع جديد باستخدام Vue CLI

الآن وبعد أن انتهينا من إعداد Vue CLI أصبح بإمكاننا الشروع باستخدامه. سنبدأ بإنشاء مشروع جديد. وبما أننا سنعمل على نقل تطبيق "مشروبات حسب" إلى Vue CLI بعد قليل، لذلك فسنسمي المشروع الجديد الذي سنعمل على إنشائه الآن بالاسم `hsoub-drinks-cli`.

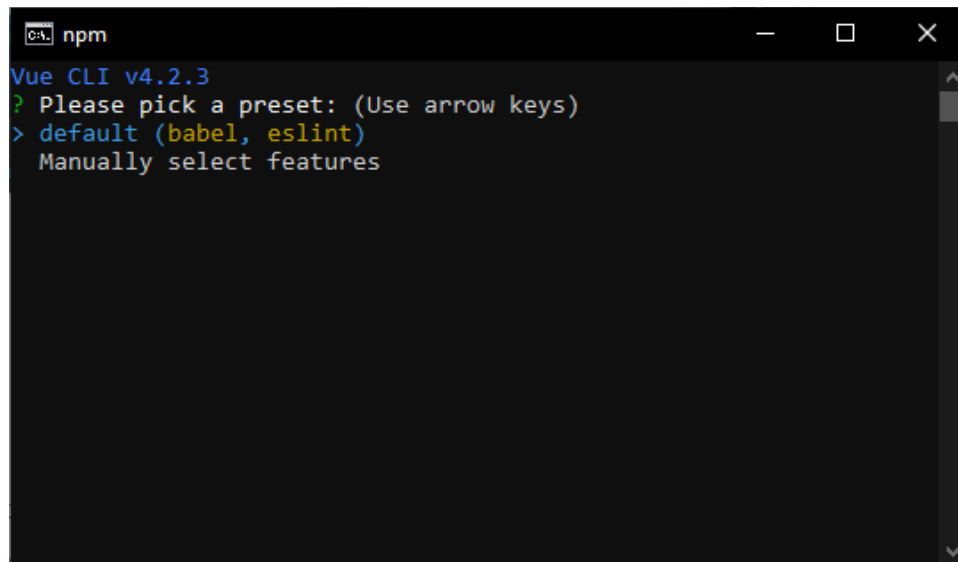
لنبدأ الآن! اكتب الأمر التالي ضمن موجه الأوامر:

```
vue create hsoub-drinks-cli
```

من السطر السابق: `vue` هو الأمر الخاص بتنفيذ Vue CLI كما أشرنا، الوسيط `create` يشير كما هو واضح إلى إنشاء مشروع جديد، أما `hsoub-drinks-cli` فهو اسم المشروع، عند تنفيذ الأمر السابق سيعمل Vue CLI على إنشاء المشروع ضمن مجلد يحمل نفس اسم المشروع، وذلك في نفس الدليل الحالي



الذي ننفذ فيه الأمر السابق. اضغط المفتاح Enter لتنفيذ الأمر، سيطلب منك Vue CLI المزيد من المعلومات قبل أن يُنشأ المشروع.



```

npm
Vue CLI v4.2.3
? Please pick a preset: (Use arrow keys)
> default (babel, eslint)
   Manually select features
  
```

من الشكل السابق، يسأل Vue CLI عن نوع القالب الذي نريد استخدامه، توجد العديد من القوالب، حيث يمكنك ضغط مفتاح السهم السفلي من لوحة المفاتيح لاختيار `Manually select features` والاطلاع على القوالب الجاهزة مسبقًا. لمشروعنا البسيط، سأختار القالب الافتراضي `default (babel, eslint)` أي اضغط على المفتاح Enter فحسب.

سيستغرق ذلك بعض الوقت لكي تُحمّل الملفات المطلوبة ويُحضّر المشروع الجديد للاستخدام. لا تقلق من عدد الملفات الكبير التي ستُنزل من الإنترنت (رغم أن المشروع بسيط).

في بعض الأحيان القليلة، يمكن أن يحدث جمود في عملية إنشاء المشروع لسبب أو لآخر. إذا شعرت أن هذه العملية قد استغرقت وقتًا طويلًا غير منطقي، أو أن عملية التحميل تقف عند حزمة محدّدة ولا تنتقل إلى حزمة تالية، عندها يمكنك ضغط `Ctrl + C` لإيقاف Vue CLI، ثم أغلق نافذة موجّه الأوامر، ثم احذف المجلّد `hsub-drinks-cli` الذي أنشأته، وأعد تشغيل الحاسوب وكرّر العملية من جديد.

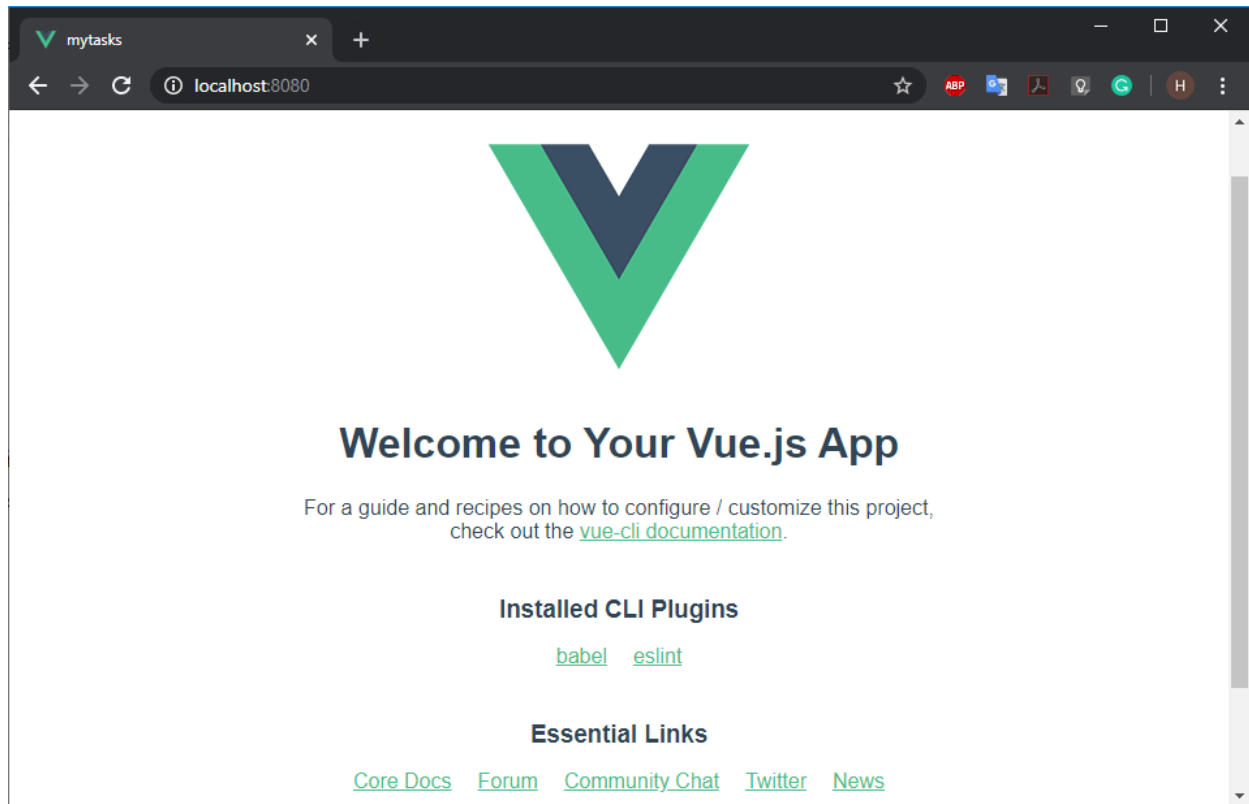
بعد الانتهاء من إنشاء المشروع. ادخل إلى المجلّد `hsub-drinks-cli` باستخدام الأمر التالي:

```
cd hsub-drinks-cli
```

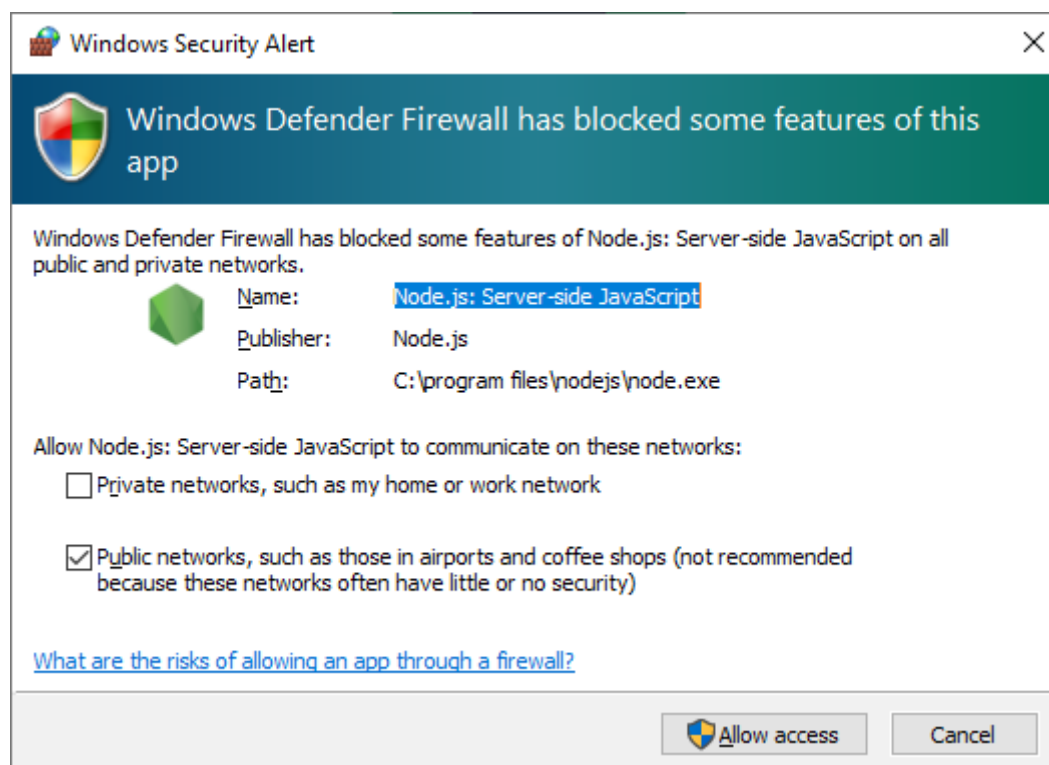
ثم ننفذ الأمر التالي لتشغيل خادم التطوير البسيط الذي يأتي مع `npm`:

```
npm run serve
```

سيؤدي ذلك إلى تشغيل هذا الخادوم على المنفذ الافتراضي 8080 ليعمل على تقديم ملفات المشروع hsub-drinks-cli الذي أنشأناه تَوَّأ. من الممكن أن تُفتح نافذة جديدة من المتصفح الافتراضي لديك بشكل تلقائي بحيث تعرض مباشرة الصفحة الرئيسية الافتراضية. أو يمكنك أن تفعل أنت ذلك بأن تفتح نافذة (أو لسان تبويب) جديدة من المتصفح لديك وتنتقل إلى العنوان التالي: <http://localhost:8080>. ستحصل على شكل شبيه بما يلي:



لاحظ أنه ممن الممكن في بعض الأحيان أن يظهر لك تنبيه أمني من نظام التشغيل، يطلب منك السماح لخادوم التطوير بالعمل. كما في الشكل التالي:

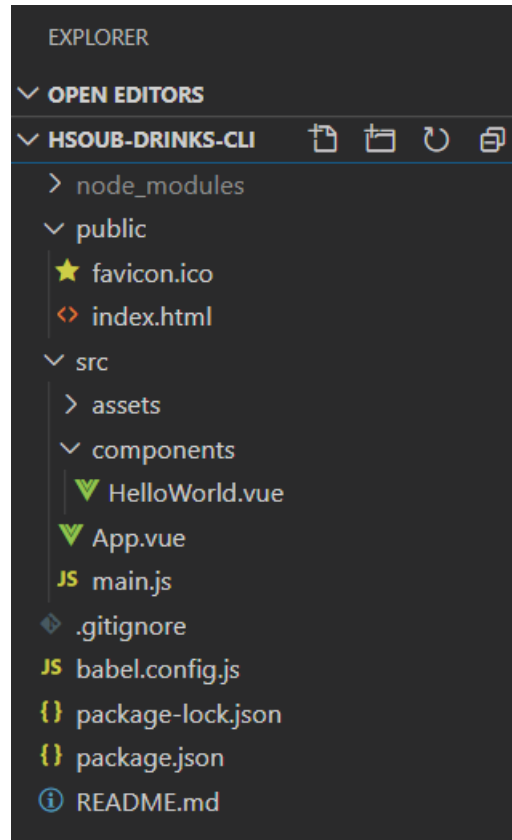


انقر الزر Allow access للسماح بذلك.

## 7.4 نظرة عامة على هيكل المشروع

انتقل الآن إلى Visual Studio Code ثم افتح مجلد المشروع الذي أنشأناه تَوًّا. يمكنك ذلك باختيار الأمر File -> Open Folder ثم انتقل إلى المجلد الذي أنشأت فيه المشروع. انقر نقرًا مزدوجًا على هذا المجلد لكي تدخل ضمنه، ثم اختر Select Folder.

سيظهر المشروع ضمن نافذة المستكشف في القسم الأيسر من الشاشة بشكل شبيه بما يلي:



سأتحدث هنا بصورة عامة عن بعض الملفات المهمة والتي ستتعامل معها بشكل متكرر. انظر الجدول التالي:

الوصف	الملف أو المجلد
عبارة عن ملف HTML عادي يرسله الخادوم إلى متصفح الويب عند زيارة الموقع: localhost:8080 من المتصفح.	<b>public/index.html</b>
يحتوي هذا المجلد على ملفات الشيفرة البرمجية، بالإضافة إلى أصول التطبيق من صور وتنسيقات وغيرها.	<b>src</b>
هذا ملف JavaScript وهو مسؤول عن إعدادات تطبيق Vue JS. كما يحتوي هذا الملف على أية حزم من طرف ثالث يمكن أن يستخدمها التطبيق	<b>src/main.js</b>
الملفات ذات الامتداد vue عمومًا، هي الملفات التي توضع ضمنها المكونات (Components)، حيث سنطبق مبدأ فصل المكونات بشكل كامل عن ملفات HTML العادية.	<b>src/App.vue</b>
يحتوي على أصول (assets) الموقع مثل الصور وشعار الموقع وملفات التنسيق وغيرها.	<b>src/assets</b>

المجلد الخاص بالمكونات التي سنكتبها للتطبيق.	components
ملف vue يحتوي على مكون تجريبي بسيط، لن نحتاج إلى هذا الملف بالطبع. حيث سنستبدل به المكونات الخاصة بتطبيق "مشروبات حسب"	components/HelloWorld.vue

احذف الملف HelloWorld.vue باختباره من نافذة المستكشف ثم ضغط الزر Delete، ثم أنشئ ملفًا جديدًا ضمن المجلد Components بالنقر بزر الفأرة الأيمن ثم اختيار الأمر New File. سمّ الملف الجديد بالاسم drink.vue. سيحتوي هذا الملف المكون الممثل لمشروب محدد (راجع الفصل السابق).

انسخ الشيفرة البرمجية التالية إلى الملف drink.vue الجديد:

```
<template>
  <div v-on:click="select" class="drink" v-bind:class="{ 'active-drink': is_selected}">
    <div class="description">
      <span class="title">{{name}}</span>
    </div>
  </div>
</template>

<script>
export default {
  name: "drink",
  props: {
    name: {
      type: String
    },
    selectedDrink: {
      type: String
    }
  },
  computed: {
    is_selected() {
      return this.selectedDrink === this.name;
    }
  },
  methods: {
    select() {
      this.$emit("drink_selected_event", this.name);
    }
  }
};
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
```

```
<style scoped>
.drinks {
  padding: 0 40px;
  margin-bottom: 40px
}

.drinks .drink {
  background-color: #fff;
  -webkit-box-shadow: 0 2px 4px 0 rgba(0, 0, 0, .1);
  box-shadow: 0 2px 4px 0 rgba(0, 0, 0, .1);
  margin-top: 1rem;
  margin-bottom: 1rem;
  border-radius: .25rem;
  -webkit-box-pack: justify;
  -ms-flex-pack: justify;
  justify-content: space-between;
  cursor: pointer;
  position: relative;
  -webkit-transition: all .3s ease;
  transition: all .3s ease
}

.drinks .drink, .drinks .drink>.weight {
  display: -webkit-box;
  display: -ms-flexbox;
  display: flex
}

.drinks .drink>.description {
  width: 100%;
  padding: 1rem;
}

.drinks .drink>.description .title {
  color: #3d4852;
  display: block;
  font-weight: 700;
  margin-bottom: .25rem;
  float: right;
}

.drinks .drink>.description .description {
  font-size: .875rem;
  font-weight: 500;
  color: #8795a1;
  line-height: 1.5
}

.drinks .drink>.price {
  width: 20%;
  color: #09848d;
  display: -webkit-box;
```

```

    display: -ms-flexbox;
    display: flex;
    padding-top: 1.5rem;
    font-family: Crimson Text, serif;
    font-weight: 600
  }

  .drinks .drink>.price .dollar-sign {
    font-size: 24px;
    font-weight: 700
  }

  .drinks .drink>.price .number {
    font-size: 72px;
    line-height: .5
  }

  .drinks .active-drink, .drinks .drink:hover {
    -webkit-box-shadow: 0 15px 30px 0 rgba(0, 0, 0, .11), 0 5px 15px 0
    rgba(0, 0, 0, .08);
    box-shadow: 0 15px 30px 0 rgba(0, 0, 0, .11), 0 5px 15px 0 rgba(0,
    0, 0, .08)
  }

  .drinks .active-drink:after, .drinks .drink:hover:after {
    border-width: 2px;
    border-color: #7dacaf;
    border-radius: .25rem;
    content: "";
    position: absolute;
    display: block;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0
  }

  .drinks .active-drink, .drinks .drink:hover {
    background-color: lightgray;
    -webkit-box-shadow: 0 15px 30px 0 rgba(0, 0, 0, .11), 0 5px 15px 0
    rgba(0, 0, 0, .08);
    box-shadow: 0 15px 30px 0 rgba(0, 0, 0, .11), 0 5px 15px 0 rgba(0,
    0, 0, .08)
  }

  .drinks .active-drink:after, .drinks .drink:hover:after {
    border-width: 2px;
    border-color: #7dacaf;
    border-radius: .25rem;
    content: "";
    position: absolute;
    display: block;
  }

```

```

    top: 0;
    bottom: 0;
    left: 0;
    right: 0
  }
</style>

```

لاحظ معي أن الملف السابق يتكوّن من الأقسام الثلاثة التالية:

```

<template>
...
</template>

<script>
...
</script>

<style>
...
</style>

```

يمكن أن يحتوي أي ملف يحمل الامتداد `vue` (وبالتالي أي مكون) على ثلاثة أقسام تُعبر عنها بالوسوم `<template>` و `<script>` و `<style>`. الوسم `<template>` إلزامي، والوسمين الآخرين اختياريين. قد تظن أن الشيفرة السابقة معقدة بعض الشيء إلا أنها بسيطة في الواقع. الشيفرة الموجودة هنا مماثلة لتلك الموجودة في الفصل السابق إلى حد كبير. حيث عملت على تجميع جميع الشيفرات البرمجية أو التنسيقات الخاصة بالمكون `drink` ضمن الملف `drink.vue`. أي أننا استطعنا عزل كل ما يتعلق بالمكون `drink` ضمن هذا الملف. وفي ذلك فائدة كبيرة تتمثل في فصل وتنظيم الشيفرة في أجزاء منطقية يسهل التعامل معها.

يحتوي القسم `<template>` على شيفرة HTML الخاصة بقالب المكون. أما القسم `<script>` فيحتوي على شيفرة `Vue.js` اللازمة للمكون. أما القسم `<style>` فكما هو واضح يحتوي على التنسيق اللازم للمكون. يمكن في بعض الأحيان (كما في حالتنا هنا) استخدام الكلمة `scoped` مع الوسم `<style>` بهدف أن نجعل التنسيق خاصًا بالمكون الحالي.

أريد التركيز على الشيفرة الموجودة ضمن القسم `<script>`. لاحظ أنني قد استخدمت التعليمة `export default`. هذه التعليمة هي تعليمة `JavaScript` ووظيفتها السماح بتصدير كائن `Vue.js` الذي يأتي بعدها إلى خارج الوحدة البرمجية (الملف) `drink.vue` وبالتالي يمكن استيراده فيما بعد باستخدام التعليمة `import` كما سنرى بعد قليل. أما كائن `Vue.js` نفسه فهو مماثل لذلك الموجود في الفصل السابق.



جاء الآن دور المكوّن `drinksselector` والذي أجريت تعديلاً طفيفاً على اسمه الذي كان في الفصل السابق وذلك لجعل الأمر أكثر سهولة في التعامل معه.

أنشئ ملفاً جديداً ضمن المجلد `Components` بالنقر بزر الفأرة الأيمن ثم اختيار الأمر `New File`. سمّ الملف الجديد بالاسم `drinksselector.vue`. سيحتوي هذا الملف المكوّن الأساسي الذي ينظّم عملية اختيار المشروبات (مكوّنات `drink`) ضمنه (راجع الفصل السابق).

انسخ الشيفرة البرمجية التالية إلى الملف `drinksselector.vue` الجديد:

```
<template>
  <div class="drinks">
    <drink
      v-for="drink in drinks"
      v-bind:key="drink.name"
      v-bind:name="drink"
      v-on:drink_selected_event="drink_selected_handler"
      v-bind:selectedDrink="current_drink"
    ></drink>
  </div>
</template>

<script>
import drink from "../drink.vue";

export default {
  name: "drinksselector",
  components: {
    drink
  },
  data() {
    return {
      drinks: ["شاي", "قهوة", "شاي أخضر", "زهورات", "با بونج"],
      current_drink: null
    };
  },
  methods: {
    drink_selected_handler(drink_name) {
      this.current_drink = drink_name;
    }
  }
};
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>

</style>
```

لاحظ معي أنَّ القسم `<style>` هنا فارغ. قد تتساءل كيف سنطبق التنسيق الخاص بهذا المكوّن كما فعلنا في الفصل السابق. في الحقيقة سيطبق التنسيق اللازم لهذا المكوّن من خلال المكوّن الابن `drink`. رغم أننا وضعنا الكلمة `scoped` ضمن القسم `<style>` من المكوّن `drink` كما مرّ معنا قبل قليل، إلّا أنَّ التنسيقات الخاصة بالمكوّن `drinksselector` ستطبق عليه. السبب في ذلك هو أنَّ المكوّن `drink` هو مكوّن متداخل (مكوّن ابن) مع المكوّن `drinksselector` فتطبق التنسيقات على المكوّن الأعلى (الأب) في هذه الحالة بشكل تلقائي.

لاحظ أيضًا السطر الذي يلي وسم الفتح `<script>`:

```
import drink from "../drink.vue";
```

تعمل هذه التعليمة البرمجية على استيراد كائن الكائن `drink` من الملف `drink.vue`. استطعنا استخدام التعليمة `import` هنا بسبب وجود التعليمة `export default` ضمن الملف `drink.vue` كما مرّ معنا قبل لحظات.

جاء الآن دور ملف التطبيق الرئيسي `App.vue` والذي يعد بحد ذاته مكوّنًا، ولكنّه المكوّن الرئيسي في تطبيق ما. افتح هذا الملف وانسخ إليه الشيفرة التالية:

```
<template>
  <div>
    <div class="header">
      <span id="logo">مشروبات حسوب</span>
    </div>

    <div id="app" class="container">
      <div class="content">
        <h1 class="title">المشروبات المتوفرة</h1>
        <drinksselector></drinksselector>
      </div>
    </div>
  </div>
</template>

<script>
import drinksselector from "../components/drinksselector.vue";

export default {
  name: "App",
  components: {
    drinksselector
  }
};
</script>
```

```
<style>
@import "../assets/styles/app.css";
</style>
```

لاحظ كيف أدرجنا المكوّن drinkselector ضمن القسم components الموجود بدوره ضمن القسم <script>. لاحظ أيضًا كيف استخدمنا التنسيقات ضمن القسم <style>. لقد ربطنا ملف التنسيقات app.css الذي سننشئه بعد قليل بهذا المكوّن عن طريق التعليمة @import.

أضف الآن مجلدًا جديدًا ضمن المجلّد src وذلك بالنقر بزر الفأرة الأيمن على المجلّد src ثم اختيار الأمر New Folder. سمّ المجلّد الجديد بالاسم assets. ثم أنشئ مجلدًا آخر ضمن المجلّد assets بنفس الأسلوب السابق وسمّه styles. أنشئ الآن ضمن المجلّد styles ملفًا اسمه app.css وانسخ إليه التنسيقات التالية:

```
@import url(//fonts.googleapis.com/earlyaccess/notonaskharabic.css);
body {
  height: 100vh;
  -webkit-font-smoothing: auto;
  -moz-osx-font-smoothing: auto;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  font-family: 'Noto Naskh Arabic', serif;
  background-color: #ccdcdd;
  background-repeat: no-repeat;
  background-position: 100% 100%
}

span#logo {
  font-weight: 700;
  color: #eee;
  font-size: larger;
  letter-spacing: .05em;
  padding-left: .5rem;
  padding-right: .5rem;
  padding-bottom: 1rem;
  float: right;
  padding-top: 6px;
  margin-right: 20px;
}

.header {
  background-color: slategray;
  width: 80%;
  height: 50px;
  margin-left: auto;
  margin-right: auto;
```

```

}

h1.title {
  text-align: center;
  font-size: 1.875rem;
  font-weight: 500;
  color: #2d3336
}

h2.subtitle {
  margin: 8px auto;
  font-size: x-large;
  text-align: center;
  line-height: 1.5;
  max-width: 500px;
  color: #5c6162
}

.content {
  margin-left: auto;
  margin-right: auto;
  padding-top: 1.5rem;
  padding-bottom: 1.5rem;
  width: 620px
}

```

انتقل الآن إلى الملف `index.html` واحرص على أن يكون محتواه مماثلاً للشفرة التالية:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-
scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %>
doesn't work properly without JavaScript enabled. Please enable it to
continue.</strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>

```

لقد انتهينا الآن من جميع الملفات! اذهب إلى المتصفح ثم انتقل إلى العنوان `http://localhost:8080` لتجد تطبيق "مشروبات حسوب" وقد عمل من جديد. ولكن هذه المرة باستخدام Vue CLI.



بعد الانتهاء من عملية التطوير بشكل كامل، يمكن استخدام الأمر التالي لبناء التطبيق وتجهيزه ضمن بيئة الإنتاج (Production Environment) للاستخدام النهائي باستخدام الأمر:

```
npm run build
```

يمكنك الاطلاع على المزيد حول هذا الموضوع من خلال [هذا الرابط](#) الذي يحيلك إلى التوثيق الرسمي باللغة الإنجليزية.

## 7.5 ختام الفصل

تعلمنا في هذا الفصل كيفية إنشاء تطبيق Vue.js متكامل باستخدام قالب جاهز منحنا إيّاه Vue CLI. تعلمنا بدايةً ما هو مفهوم Vue CLI وكيفية إعدادهِ، وكيف ننشئ مشروع جديد باستخدامهِ.

كما عملنا على تحويل كامل تطبيق "مشروبات حسوب" من الأسلوب القديم والمفيد في حال كُنّا نريد تجربة مزايا جديدة، إلى أسلوب Vue CLI بالكامل.

بهذا الفصل نكون قد خطونا خطوة مهمة ومتقدمة في العمل مع Vue.js.

# 8. التعامل مع دخل المستخدم عن

## طريق نماذج الإدخال

سنتعلم في هذا الفصل:

- بناء هيكل تطبيق بسيط لشرح أفكار الفصل
- استخدام إطار العمل Bootstrap
- استخدام بنى معطيات متقدمة مع عناصر الإدخال النصية
- المعدّلات (Modifiers) في Vue.js
- التعامل مع مربعات الاختيار (Checkboxes) وأزرار الانتقاء (Radiobuttons)
- التعامل مع القائمة المنسدلة `<select>`
- إرسال البيانات

لقد تعاملنا في بداية الكتاب مع عناصر الإدخال العادية التي يستخدمها المستخدم لإدخال البيانات ضمن صفحات الويب. سنعمل في هذا الفصل على التوسع في التعامل مع هذه العناصر بالإضافة إلى الاطلاع على طريقة التعامل مع عناصر إدخال HTML جديدة. كما سنتعلم كيفية التعامل مع أطر عمل CSS جاهزة، حيث لم يسبق لنا التعامل معها مسبقًا.

## 8.1 بناء هيكل تطبيق بسيط

كما جرت العادة، سنبنّي تطبيق بسيط من أجل هذا الفصل بهدف تطبيق الأفكار الواردة فيه. سيتكوّن تطبيقنا من صفحة واحدة فقط تحوي عدد من عناصر HTML التي ننوي التعامل معها. أنشئ مشروع جديد وسّمه input-forms-cli كما فعلنا مسبقاً في الفصل السابق (راجع الفقرة "بناء هيكل تطبيق بسيط لشرح أفكار الفصل"). بعد الانتهاء من عملية الإنشاء، استخدم Visual Studio Code في فتح المشروع (المجلّد) الذي أنشأته تَوّاً.

لن نحتاج سوى ملف مكوّن واحد وهو App.vue لذلك احذف الملف HelloWorld.vue كما فعلنا في الفصل السابق. ثم افتح الملف App.vue واستبدل المحتوى الحالي بالشفيرة البرمجية التالية:

```
<template>
  <div class="container">
    <form>
      <div class="row">
        <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-offset-3">
          <h1 class="text-right">الملف الشخصي للمستخدم</h1>
          <hr>
          <div class="form-group">
            <label class="float-right" for="firstname">
الاسم</label>

            <input
              type="text"
              id="firstname"
              class="form-control"
              v-model="userMainData.firstname">
          </div>
          <div class="form-group">
            <label class="float-right" for="lastname">
الكنية</label>

            <input
              type="text"
              id="lastname"
              class="form-control"
              v-model="userMainData.lastname">
          </div>
          <div class="form-group">
            <label class="float-right" for="age">
العمر</label>

            <input
              type="number"
              id="age"
              class="form-control"
              v-model="userMainData.age">
          </div>
        </div>
      </div>
    </form>
  </div>
</template>
```

```

        </div>
        <div class="form-group">
          <label class="float-right" for="password">كلمة
الم المرور</label>
          <input
            type="password"
            id="password"
            class="form-control"
            v-model="userMainData.password">
          </div>
        </div>
      </div>
      <div class="row">
        <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-
6 col-md-offset-3 form-group">
          <label class="float-right" for="description">
نبذة</label><br>
          <textarea
            id="description"
            rows="5"
            class="form-control"
            v-model="description"></textarea>
          </div>
        </div>
        <div class="row">
          <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-
6 col-md-offset-3">
            <div class="form-group">
              <label class="float-right" for="graduate">
                <input
                  type="checkbox"
                  id="graduate"
                  value="متخرج"
                  v-model="status"> متخرج
              </label>
              <label class="float-right" for="smoker">
                <input
                  type="checkbox"
                  id="working"
                  value="أعمل حاليًا"
                  v-model="status"> أعمل حاليًا
              </label>
            </div>
          </div>
        </div>
      </div>
      <div class="row">
        <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-
6 col-md-offset-3 form-group">
          <label class="float-right" for="male">
            <input
              type="radio"
              id="male"

```



```

        value="ذكر"
        v-model="gender"> <span>ذكر</span>
    </label>
    <label class="float-right" for="female">
        <input
            type="radio"
            id="female"
            value="أنثى"
            v-model="gender"> <span>أنثى</span>
    </label>
</div>
</div>
<div class="row">
    <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-
6 col-md-offset-3 from-group">
        <label class="float-right" for="subscriptionKind">
نوع الاشتراك</label>
        <select
            id="subscriptionKind"
            class="form-control"
            v-model="selectedSubscription">

            <option v-for="kind in subscriptionKinds" v-
bind:key="kind">
                {{ kind }}
            </option>
        </select>
    </div>
</div>
<hr>
<div class="row">
    <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-
1">
        <button
            class="btn btn-primary">إرسال</button>
    </div>
</div>
</form>
<hr>
<div class="row">
    <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6
col-md-offset-3">
        <div class="card card-info">
            <div class="card-header text-right">
                <h4>البيانات المُدخلة</h4>
            </div>
            <div class="card-body">
                <div class="card-text text-right">
                    <p>الاسم: {{ userMainData.firstname }}</p>
                    <p>الكنية: {{ userMainData.lastname }}</p>
                    <p>العمر: {{ userMainData.age }}</p>
                </div>
            </div>
        </div>
    </div>
</div>

```

أضف مجلدًا جديدًا ضمن المجلد assets وسمّه styles ثم أضف ملفًا جديدًا ضمن المجلد الأخير وسمّه app.css كما فعلنا في الفصل السابق.

```
@import url(//fonts.googleapis.com/earlyaccess/notonaskharabic.css);

body{
    font-family: 'Noto Naskh Arabic', serif;
```

}

احفظ جميع التعديلات، ثم افتح موجّه الأوامر في ويندوز، وبعدها انتقل إلى مجلد التطبيق input-forms-cli الذي أنشأته قبل قليل، ونفّذ الأمر:

npm run serve

يعمل هذا الأمر كما نعلم على تشغيل خادم الويب الخاص بالتطوير، اذهب الآن إلى متصفح الويب لديك، وانتقل إلى العنوان <http://localhost:8080> ليظهر لك شكل شبيه بما يلي:

## الملف الشخصي للمستخدم

الاسم	<input type="text"/>
الكنية	<input type="text"/>
العمر	<input type="text" value="0"/>
كلمة المرور	<input type="password"/>
نبذة	<div>اكتب نبذة قصيرة عنك!</div> <div></div>
<input type="checkbox"/> متخرج <input type="checkbox"/> أعمل حالياً	
<input checked="" type="radio"/> ذكر <input type="radio"/> أنثى	
نوع الاشتراك	<div>فضي</div> <div>▼</div>
<input type="button" value="إرسال"/>	

### البيانات المدخلة

الاسم:

الكنية:

العمر: 0

كلمة المرور:

نبذة: اكتب نبذة قصيرة عنك!

الوضع الحالي

النوع: ذكر

نوع الاشتراك: فضي

## 8.2 استخدام إطار العمل Bootstrap

ربما تكون قد استغربت قليلاً من عدم استخدامنا لأي مكتبة أو إطار عمل CSS في عمليات التنسيق التي كنا نجريها في تطبيقاتنا السابقة. بدلاً من ذلك، كنا نستخدم شيفرة CSS عادية فحسب في تنسيق التطبيقات. يعود سبب ذلك إلى أنه لا يُنصح أبداً إضافة إطار عمل CSS بالطريقة التقليدية التي تتمثل في استخدام الوسم `<link>` كما يفعل أغلبنا عند تطوير تطبيقات أمامية (Frontend Applications). يعود السبب في ذلك لأن أغلب أطر عمل CSS تحتوي على شيفرة JavaScript قد لا تكون متوافقة مع Vue.js. علينا إذاً استخدام إطار عمل CSS نضمن أن يكون متوافقاً مع Vue.js لكي لا تحصل مفاجآت غير مرغوبة! سأحدث هنا عن استخدام إطار العمل الشهير Bootstrap لكي نُكسب تطبيقاتنا تنسيقات جميلة وقوية وبشكل متوافق مع Vue.js.

افتح نافذة موجه الأوامر في ويندوز، ثم نَقِّد الأمر التالي:

```
npm install bootstrap@4.0.0
```

بعد الانتهاء وعودة موجه الأوامر إلى حالته الطبيعية، أضف السطر التالي إلى الملف `main.js`:

```
import "bootstrap/dist/css/bootstrap.min.css";
```

ستصبح محتويات الملف `main.js` مشابهة لما يلي:

```
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

import "bootstrap/dist/css/bootstrap.min.css";

new Vue({
  render: h => h(App),
}).$mount('#app')
```

نكون بهذا الشكل قد نصبنا إطار العمل Bootstrap ومن ثم أضفناه إلى التطبيق الخاص بنا ليصبح جاهزاً للاستخدام. وهكذا نكون قد استغفينا عن كتابة تنسيقات مخصصة إلى حد كبير.

## 8.3 استخدام بنى معطيات متقدمة مع عناصر الإدخال النصية

في الحقيقة رغم أن عنوان هذه الفقرة مغرٍ بعض الشيء، إلا أننا في الواقع سنستخدم كائن JavaScript عادي لتمثيل بيانات المستخدم الأساسية ضمن تطبيقنا الحالي. إذا راجعت شيفرة التطبيق الأساسية التي أوردتها في الفقرة الأولى، وتحديدًا ضمن القسم `<script>` ستجد البنية البسيطة التالية:

```
userMainData:{
  firstname:'',
  lastname:'',
  age:0,
  password:'',
}
```

كما أوضحت، سيحمل الحقل `userMainData` بيانات المستخدم الرئيسية التالية على الترتيب: الاسم، الكنية، العمر، كلمة المرور.

السؤال هنا، كيف سنربط عناصر HTML الموافقة؟ الجواب بسيط، سنستخدم `v-model` للربط ثنائي الاتجاه. انظر إلى الشيفرة المقتطعة من الشيفرة الموجودة ضمن القسم `<template>`:

```
<div class="form-group">
  <label class="float-right" for="firstname">الاسم</label>
  <input
    type="text"
    id="firstname"
    class="form-control"
    v-model="userMainData.firstname">
</div>
<div class="form-group">
  <label class="float-right" for="lastname">الكنية</label>
  <input
    type="text"
    id="lastname"
    class="form-control"
    v-model="userMainData.lastname">
</div>
<div class="form-group">
  <label class="float-right" for="age">العمر</label>
  <input
    type="number"
    id="age"
    class="form-control"
    v-model="userMainData.age">
</div>
<div class="form-group">
  <label class="float-right" for="password">كلمة المرور</label>
  <input
```

```

type="password"
id="password"
class="form-control"
v-model="userMainData.password">
</div>

```

لاحظ معي كيف استخدمت الحقل `userMainData` لربط كل خاصية من خصائصه بعنصر HTML الموافق. انظر مثلاً ماذا استخدمت لربط حقل الاسم `firstname`:

```
userMainData.firstname
```

لقد استخدمت النقطة للفصل بين `userMainData` وبين `firstname` وهذا جائز تمامًا! يمكنك مراجعة باقي عناصر HTML لترى كيف استخدمت نفس الترميز السابق. الفائدة في استخدام هذا الأسلوب هي في تنظيم وتمثيل البيانات بشكل منطقي في التطبيق.

حاول كتابة أي شيء ضمن الحقول الأربعة السابقة، ستجد أن ذلك سينعكس مباشرة على القسم السفلي (البيانات المُدخلة) من الصفحة، والذي جعلته خصيصًا لمشاهدة النتائج التي سنجرّيها على البيانات في القسم العلوي.

في الحقيقة لقد تعاملنا مسبقًا مع عناصر الإدخال العادية فيما سبق من فصول، وسنتعامل مع أنواع أخرى من عناصر الإدخال في هذا الفصل. أحد العناصر الجديدة التي سنتعامل معها هو العنصر `<textarea>` والذي نتعامل معه كما نتعامل مع عناصر الإدخال العادية تمامًا. إذ وضعت الموجه `v-model` ضمن هذا العنصر ليرتبط مع الحقل `description` كما هو واضح من الشيفرة الواردة في الفقرة الأولى من هذا الفصل.

## 8.4 المعدلات (Modifiers) في Vue.js

يحتاج المبرمج في بعض الأحيان إلى تعديل سلوك الاستجابة لـ `Vue.js`. فمثلاً إذا لاحظت في تطبيقنا هذا، أن أي شيء تكتبه ضمن أي عنصر من عناصر الإدخال النصية، سينعكس مباشرة ضمن القسم السفلي. قد يكون هذا السلوك غير مرغوب أحياناً، فقد ترغب ربما بأن لا يستجيب `Vue.js` مباشرة لما يكتبه المستخدم، بل بأن تؤجل هذه الاستجابة حتى ينتهي المستخدم مما يكتبه وينتقل إلى حقل نصي آخر مثلاً.

السيناريو السابق ممكن تمامًا باستخدام تقنية المعدلات (modifiers) حيث يمكن أن نستخدم المعدل `lazy` لكي نخبر `Vue.js` بأن يؤجل الاستجابة لأي عنصر إدخال مرتبط معه ريثما يعمل المستخدم على مغادرة هذا العنصر (يفقده التركيز Focus). انتقل إلى الشيفرة البرمجية الخاصة بعنصر الإدخال `firstname` ضمن `<template>` ثم أضف المعدل `lazy` إلى الموجه `v-model` على الشكل التالي:

```
v-model.lazy="userMainData.firstname"
```

احفظ التغييرات، ثم حاول كتابة شيء ما ضمن الحقل `firstname` ولا تحاول مغادرة الحقل، ستجد هذه المرة أن ذلك لن ينعكس مباشرة ضمن القسم السفلي. غادر الآن ذلك الحقل إلى حقل آخر، ستجد أن بيانات الاسم قد ظهرت دفعة واحدة في القسم السفلي.

توجد معدّلات أخرى مثل `number` الذي يُستخدم لكي يُفسّر دخل المستخدم على أنه رقم بدلاً من النص. وهناك أيضًا المعدّل `trim` الذي يُستخدم للتخلص من المحارف الفارغة على يمين ويسار النص المُدخّل. يمكن استخدام أي معدّل كما استخدمنا المعدّل `lazy` قبل قليل، كما ويمكن استخدامها بشكل مركّب. انظر مثلاً كيف يمكن استخدام المعدلين `lazy` و `trim` بشكل مركّب:

```
v-model.lazy.trim = "userMainData.firstname"
```

## 8.5 التعامل مع مربعات الاختيار وأزرار الانتقاء

سنتعلم في هذه الفقرة كيفية التعامل مع مربع الاختيار (Checkbox) وزر الانتقاء (Radiobutton). سنبدأ أولاً مع مربع الاختيار.

في تطبيقنا هذا، سنستخدم مربعي اختيار للتعبير عن كون المستخدم متخرّج أم غير متخرّج. بهدف التعامل مع هذين المربعين، عزّفت حقلاً جديداً اسمه `status` على أنه مصفوفة (انظر القسم `<script>`). ربطت هذه المصفوفة مع مربعي الاختيار بإسنادها إلى الموجّه `v-model` لكل من المربعين. انظر الشيفرة المقتطعة من القسم `<template>`:

```
<label class="float-right" for="graduate">
  <input
    type="checkbox"
    id="graduate"
    value="متخرج"
    v-model="status"> متخرج
</label>
<label class="float-right" for="smoker">
  <input
    type="checkbox"
    id="working"
    value="أعمل حالياً"
    v-model="status"> أعمل حالياً
</label>
```

لاحظ كيف أننا ربطنا مربعي الاختيار بنفس الشكل. الذي سيحدث وراء الكواليس، هو أنه عندما يختار المستخدم أحد المربعين سيعمل Vue.js على إدراج قيمة الحقل الذي تم اختياره كعنصر في

المصفوفة `status`، فإذا اختار المستخدم كلا المربعين، فسيُدْرَج `Vue.js` عنصرين ضمن المصفوفة، يعتبر كل منهما عن قيمة مربع الاختيار. أي أن عدد عناصر المصفوفة سيكون مساوياً لعدد المربعات التي اختارها المستخدم، طالما أن هذه المربعات قد تم ربطها بنفس الشكل.

يمكن تطبيق نفس المبدأ تقريباً على أضرار الانتقاء، في تطبيقنا هذا لدينا زري انتقاء يُعبران عن نوع المستخدم (ذكر أم أنثى). عرّفت حقلاً جديداً أسميته `gender` يحمل القيمة الافتراضية ذكر. انظر معي إلى الشيفرة المقتطعة من القسم `<template>` لزري الانتقاء:

```
<label class="float-right" for="male">
  <input
    type="radio"
    id="male"
    value="ذكر"
    v-model="gender"> <span>ذكر</span>
</label>
<label class="float-right" for="female">
  <input
    type="radio"
    id="female"
    value="أنثى"
    v-model="gender"> <span>أنثى</span>
</label>
```

لاحظ معي بدايةً أنني قد وضعت القيمة `value` لكل من الزرين السابقين لتكونا "ذكر" و "أنثى" على الترتيب. لاحظ أيضاً كيف استخدمت نفس أسلوب الربط باستخدام الموجه `v-model` لكل من هذين الزرين. الذي سيحدث عند بدء تشغيل التطبيق أن الحقل `gender` سيحمل القيمة ذكر بشكل افتراضي كما ذكرت قبل قليل، وبالتالي سيختار `Vue.js` الزر المعبّر عن قيمة "ذكر" لأن قيمته ستكون مماثلة لقيمة الحقل `gender` في هذه الحالة. وهذا ما يبرز الاختيار الافتراضي لهذا الزر عند البدء بتشغيل التطبيق.

جرب أن تجري الآن بعض التجارب على مربعي الاختيار وزري الانتقاء، ولاحظ النتائج التي ستحدث ضمن القسم السفلي المخصّص لعرض البيانات.

## 8.6 التعامل مع القائمة المنسدلة

بقي لنا أن نتعلّم كيفية الربط مع القائمة المنسدلة `<select>`. سينقسم عملنا هنا إلى قسمين. الأول هو تعبئة هذه القائمة، والثاني هو الربط مع `Vue.js` باستخدام الموجه `v-model`.



بالنسبة لتعبئة هذه القائمة بالبيانات الأولية التي سيختار منها المستخدم، فقد عرّفت الحقل `subscriptionKinds` وهو على شكل مصفوفة تحوي العناصر التالية:

```
subscriptionKinds: ['ذهبي', 'فضي', 'عادي']
```

تعتبر هذه المصفوفة عن نوع الاشتراك الذي يرغب المستخدم باعتماده. سيكون لدينا كما هو واضح ثلاثة اشتراكات. الشيفرة البرمجية المسؤولة عن تعبئة عنصر القائمة هي التالية:

```
<option v-for="kind in subscriptionKinds" v-bind:key="kind">
  {{ kind }}
</option>
```

هذه الشيفرة مأخوذة من القسم `<template>` بالطبع. وهي عبارة عن حلقة بسيطة تعمل على تعبئة القائمة من خلال توليد عناصر `<option>` العمود الفقري لعنصر القائمة `<select>`.

بالنسبة لعملية الربط فقد عرّفت حقلًا آخرًا أسميته `selectedSubscription` وأسندت له القيمة الافتراضية "فضي". لاحظ معي أنّ هذه القيمة مماثلة للعنصر الثاني من عناصر المصفوفة `subscriptionKinds`. بعد ذلك، ربطت عنصر القائمة باستخدام `v-model` على النحو التالي:

```
v-model="selectedSubscription"
```

وهكذا وعند تشغيل التطبيق للمرة الأولى، سيظهر العنصر الثاني فضي وقد اختير بشكل افتراضي. جرب الآن أن تغير خياراتك ضمن القائمة وانظر كيف سينعكس ذلك ضمن القسم السفلي المخصّص لعرض البيانات.

## 8.7 إرسال البيانات

كما هو معلوم، عند وضع عنصر الزر `button` ضمن عنصر النموذج `form`، فسيؤدي نقر هذا الزر إلى إرسال بيانات النموذج (`form`) إلى الخادوم. في تطبيقنا البسيط هذا، وضعنا زرًا لإرسال البيانات كما تعلم، ولكن بما أنّه ليس لدينا حاليًا أي تطبيق يعمل على الخادوم لمعالجة لبيانات المرسلّة، لذلك سنعمل على معالجة البيانات محليًا بشكل وهمي، لذلك سنغيّر من سلوك هذا الزر لكي نمنعه من التصرف بالشكل الافتراضي.

سنستخدم لهذه الغاية المعدّل `prevent` مع الموجه `v-on:click`. اعمل على تعديل شيفرة HTML الخاصة بزر الإرسال لتصبح على النحو التالي:

```
<button
  v-on:click.prevent="submitInfo"
  class="btn btn-primary">إرسال
</button>
```

سنضيف القسم `method` لكي نتمكن من تعريف التابع `submitInfo`. سأضع هنا كامل

قسم `<script>` بعد التعديل:

```
methods: {
  submitInfo() {
    alert("تمت معالجة البيانات");
  }
}
```

وضعت رسالة بسيطة تشير إلى أن البيانات قد تمت معالجتها بشكل افتراضي محليًا. يمكن أن تضع بدلاً من هذه الرسالة أن شيفرة قد تجدها مناسبة للبيانات التي أدخلها المستخدم. الهدف هنا هو أن تفهم المبدأ بحيث يمكنك تكييفه فيما بعد بحسب احتياجاتك.

## 8.8 ختام الفصل

تعلمنا في هذا الفصل كيفية استخدام أطر عمل جاهزة لتنسيق المحتوى، حيث استخدمنا في هذا الفصل إطار العمل الشهير Bootstrap. كما تعلمنا كيف نتعامل مع عدد من عناصر HTML المخصصة لاستقبال الدخل من المستخدم، حيث تعاملنا مع عناصر الإدخال النصية البسيطة بالإضافة إلى عنصر الإدخال ذو الأسطر المتعددة `<textarea>` وأيضًا مربعات الاختيار وأزرار الانتقاء وعنصر القائمة المنسدلة.

في الحقيقة، يمكن أن تبني عناصر إدخال مخصصة بك وفق احتياجاتك الخاصة وذلك باستخدام المكونات كما مر معنا في فصول سابقة. أرجو لك الفائدة من هذا الفصل، أراك في الفصول القادمة إن شاء الله.

## 9. المرشحات Filter والمخاليط Mixin

سنتعلم في هذا الفصل:

- المرشح (Filter)
- المخلوط (Mixin)

سنتابع عملنا في هذا الفصل مع ميزتين مفيدتين في Vue.js وهما: المرشحات (Filters) والمخاليط (Mixins). نعتبر هاتين الميزتين على بساطتهما من المزايا المتقدمة نسبيًا في Vue.js. سنوضح المقصود بكل منهما في هذا الفصل، وذلك بكتابة مثالين تطبيقيين بسيطين، لتوضيح الغاية من استخدام هاتين التقنيتين بشكل جيد.

## 9.1 المرشح (Filter)

المرشح (Filter) هو ميزة مفيدة في Vue.js يمكن من خلالها تطبيق إجراء معين على قيمة حقل ما، بحيث يطرأ تعديل على شكله النهائي عند عرضه للمستخدم.

يمكن تعريف المرشحات التي نرغب بكتابتها ضمن قسم خاص ضمن كائن Vue.js اسمه `filters`. فالمرشح في الحقيقة عبارة عن مجرّد تابع يقبل وسيطاً واحداً، ويرجع قيمة بعد التعديل.

كمثال على ذلك، يمكن استخدام مرشح يعمل على تحويل حالة الأحرف إلى كبيرة (بالنسبة للأحرف الانجليزية)، رغم أنّها قد تكون ضمن الحقل عبارة عن مزيج بين أحرف صغيرة وكبيرة. ربما تكتشف بعد قليل عدداً كبيراً من الأمثلة حول الإمكانيات التي قد توفرها المرشحات. ولكن تذكر، أنّه من الأفضل دوماً أن تكون المعالجة البرمجية ضمن المرشحات بسيطة وسريعة. فلم تصفّم المرشحات لإجراء عمليات معقدة على البيانات تكلف الكثير من إمكانيات المعالجة. فإذا شعرت مثلاً أنّ الأمر يتجه إلى مزيد من التعقيد، فأنصحك بالتفكير باستخدام الخصائص المحسوبة بدلاً من المرشحات.

لنتناول الآن مثلاً برمجيّاً يوضّح كيفية كتابة المرشحات. أنشئ مشروعاً جديداً باستخدام الأمر `vue` من موجه الأوامر وسقّه `vue-filters` كما يلي:

```
vue create vue-filters
```

ثم افتح المشروع باستخدام Visual Studio Code. احذف الملف `HelloWorld.vue` ثم استبدل بما يلي محتويات الملف `App.vue`:

```
<template>
  <div id="app">
    <h1>Filters</h1>
    {{text | toUpper}}
  </div>
</template>

<script>
export default {
  data(){
    return {
      text:"We will apply a filter on this field."
    }
  },
  filters:{
    toUpper(value){
      return value.toUpperCase();
    }
  }
}
```

```

    }
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>

```

انظر إلى القسم الجديد `filters` كيف عرفت المرشح `toUpper` ضمنه على شكل تابع يقبل قيمة وحيدة `value`. لاحظ كيف يُرجع هذا المرشح حالة الأحرف الكبيرة لهذه القيمة عن طريق تابع `JavaScript` وهو `toUpperCase`.

لاحظ معي الآن طريقة تطبيق المرشح كما هو ظاهر من الشيفرة السابقة. نضع المحرف | بعد اسم الحقل المراد تطبيق المرشح عليه، ثم نكتب اسم المرشح. أي كما في التالي:

```
text | toUpper
```

وهذا كل شيء. ينبغي الانتباه هنا إلى أن المرشح لن يؤثر على القيمة الأصلية للحقل `text` إنما سيجري التعديل على الخرج الذي سيظهر للمستخدم فحسب. جرب الآن معاينة التطبيق البسيط السابق لترى كيف أصبح النص يظهر بأحرف طباعية كبيرة.

الأسلوب السابق في تعريف المرشحات كقسم ضمن كائن `Vue.js` يُعتبر تعريفاً محلياً (Local). يمكن تعريف المرشحات بشكل عام (Global) وذلك عن طريق استخدام التابع `Vue.filter` ضمن الملف `main.js` على الشكل التالي:

```

import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

Vue.filter('toLower', function(value){
  return value.toLowerCase();
})

new Vue({
  render: h => h(App),

```

```
}).$mount('#app')
```

كما هو واضح، يُمرّر وسيطان إلى التابع `filter` الأول هو اسم المرشح العام، أما الثاني فهو التابع الذي يحتوي على الشيفرة البرمجية الخاصة بالمرشح. المرشح هنا هو `toLowerCase` وسنجعله مسؤول عن تحويل الأحرف إلى الحالة الطباعية الصغيرة.

سنعمل الآن على تطبيق المرشح العام الجديد ضمن الملف `App.vue` وذلك على المرشح المحلي القديم على النحو التالي:

```
{{text | toUpper | toLower}}
```

لاحظ كيف طبقنا المرشح الجديد بشكل متسلسل على المرشح القديم. الذي سيحدث الآن، أن محتويات الحقل `text` سيطبق عليها المرشح `toUpper` أولاً، مما سيعطينا حالة أحرف طباعية كبيرة، ثم سيتم تطبيق المرشح `toLowerCase` على النتيجة الأخيرة، مما سيؤدي إلى تحويل حالة الأحرف إلى الحالة الطباعية الصغيرة. كان يمكن بالطبع تطبيق المرشح `toLowerCase` فقط دون المرشح `toUpper` ولكنني فضلت استخدامها بالشكل المتسلسل السابق لتوضيح إمكانية تطبيق أكثر من مرشح بنفس الوقت بشكل متسلسل كما فعلنا قبل قليل.

## 9.2 المخلوط (Mixin)

قد تُضطر في بعض الأحيان إلى كتابة مكونات تشترك بنفس الجزء من الشيفرة البرمجية تقريباً. رغم أن ذلك لا يعد مشكلة برمجية أساساً إلا أنه يشكل عادة برمجية غير جيدة. دائماً ما نسعى في البرمجة إلى تجنب تكرار الشيفرة البرمجية كما تعلم.

جاء المخلوط (Mixin) ليوجد حلاً لهذه المسألة، حيث من الممكن جعل ذلك الجزء المشترك من المكونات ضمنه بحيث تتشارك تلك المكونات بالمخلوط مما يلغي الحاجة إلى تكرار الشيفرة البرمجية.

سنحتاج بالتأكيد إلى مثال عملي يوضح هذه الفكرة. أنشئ مشروعاً جديداً وسمه `vue-mixins`، ثم افتح المشروع باستخدام Visual Studio Code. احذف الملف `HelloWorld.vue`، ثم أضف ملفين ضمن المجلد `components`، سمهما على النحو التالي: `BMI.vue` و `ImperialConverter.vue`. وأضف أيضاً ملفاً اسمه `ConverterMixin.js` إلى المجلد `src`.

فكرة هذا التطبيق هي إنشاء مكونين الأول هو `ImperialConverter` يوفر إمكانية التحويل من الكيلوغرام والسنتيمتر إلى الباوند والبوصة على الترتيب. أما المكون الثاني فهو `BMI` ووظيفته إعطاء تقرير

عن حالة مؤشر كتلة الجسم (Body Mass Index). يشترك كل من المكونين السابقين بجزء من الشيفرة البرمجية موضوع ضمن المخلوط ConverterMixin. مع العلم أنَّ كل من المكونين السابقين مستقلين تمامًا فيما يتعلق بالبيانات.

يحتوي المخلوط ConverterMixin على الشيفرة البرمجية المسؤولة عن التحويل من الكيلوغرام إلى الباوند، ومن السنتيمتر إلى البوصة. انظر محتويات الملف ConverterMixin.js:

```
export const ConverterMixin = {
  data() {
    return {
      kg_value: 0,
      cm_value: 0
    };
  },
  computed: {
    to_pounds: function () {
      return this.kg_value * 2.20462;
    },
    to_inches: function () {
      return this.cm_value * 0.393701;
    }
  }
}
```

الشيفرة البرمجية الموجودة في المخلوط بسيطة جدًا، حيث تحتوي على حقلي بيانات kg\_value و cm\_value بالإضافة إلى خاصيتين محسوبتين: to\_pounds و to\_inches.

بالنسبة للمكون ImperialConverter فهو بسيط أيضًا، انظر للشيفرة البرمجية الموجودة ضمنه:

```
<template>
  <div>
    <h2>Kilograms to Pounds Converter</h2>
    <div>
      <input type="text" style="width:50px;text-align:center;" v-
model="kg_value" />
      <span style="margin-left:8px;">Kg = {{to_pounds}} pounds.</span>
    </div>
    <br />
    <div>
      <input type="text" style="width:50px;text-align:center;" v-
model="cm_value" />
      <span style="margin-left:8px;">CM = {{to_inches}} Inches.</span>
    </div>
  </div>
</template>
```

```
<script>
import { ConverterMixin } from "../ConverterMixin";

export default {
  name: "ImperialConverter",
  mixins: [ConverterMixin],
};
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
</style>
```

يوفر هذا المكوّن عناصر HTML مناسبة لإدخال البيانات المراد تحويلها وذلك ضمن القسم `<template>`، أمّا بالنسبة للقسم `<script>` فنعمل على استيراد المخلوط باستخدام التعليمة `import` ثم نُخبر Vue.js أننا نريد استخدام هذا المخلوط عن طريق القسم `mixins`. لاحظ أنّه لا توجد شيفرة برمجية فعلية ضمن المكون لأنّه أغلب المنطق الحسابي يحدث ضمن المخلوط في مثالنا البسيط هذا.

أمّا بالنسبة للمكوّن BMI فهو المكوّن المسؤول (كما أشرنا) على حساب مؤشر كتلة الجسم بالإضافة إلى عرض تقرير بسيط للمستخدم حول القيمة المحسوبة. انظر إلى الشيفرة البرمجية لهذا المكون:

```
<template>
  <div>
    <h2>Body Mass Index (BMI)</h2>
    <div>
      <div>
        <span>Weight (Kg):</span>
        <input type="text" style="width:50px;text-align:center;" v-model="kg_value" />
      </div>
      <br />
      <div>
        <span>Height (Cm):</span>
        <input type="text" style="width:50px;text-align:center;" v-model="cm_value" />
      </div>
      <br />
      <div>
        <span>BMI value: {{bmi_value}}</span> -
        <span>{{result}}</span>
      </div>
    </div>
  </div>
</template>

<script>
import { ConverterMixin } from "../ConverterMixin";
```



```
export default {
  name: "BMI",
  mixins: [ConverterMixin],
  computed: {
    bmi_value: function () {
      return (703 * this.to_pounds) / (this.to_inches *
this.to_inches);
    },
    result: function () {
      if (this.bmi_value < 16) {
        return "Severe Thinness";
      } else if (this.bmi_value < 17) {
        return "Moderate Thinness";
      } else if (this.bmi_value < 18.5) {
        return "Mild Thinness";
      } else if (this.bmi_value < 25) {
        return "Normal";
      } else if (this.bmi_value < 30) {
        return "Overweight";
      } else if (this.bmi_value < 35) {
        return "Obese Class I";
      } else if (this.bmi_value < 40) {
        return "Obese Class II";
      } else if (this.bmi_value >= 40) {
        return "Obese Class III";
      } else {
        return "Not defined!";
      }
    },
  },
};
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
</style>
```

مرة أخرى، يوفّر القسم `<template>` واجهة المستخدم، أما القسم `<script>` فنستورد عن طريقه المخلوط `ConverterMixin` ونصرح عن استخدامه كما فعلنا تمامًا مع المكون `ImperialConverter` قبل قليل.

بقي أخيرًا الملف `App.vue` الذي سنستخدم من خلاله المكونات السابقين. انظر إلى الشيفرة البرمجية لهذا الملف:

```
<template>
  <div id="app">
    
```

```

    <imperial-converter/>
    <BMI/>
  </div>
</template>

<script>
import ImperialConverter from './components/ImperialConverter.vue'
import BMI from './components/BMI.vue'

export default {
  name: 'App',
  components: {
    ImperialConverter,
    BMI
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>

```

بعد تشغيل التطبيق ستحصل على واجهة شبيهة بما يلي:



## Imperial Converter

Kg = 264.5544 pounds.

CM = 72.440984 Inches.

## Body Mass Index (BMI)

Weight (Kg):

Height (Cm):

BMI value: 35.44064622400668 - Obese Class II

## 9.3 ختام الفصل

تحدثنا في هذا الفصل عن ميزتين جديدتين من مزايا Vue.js، وهما: المرشحات (Filters) والمخاليط (Mixins)، حيث قد وجدنا أنَّ المرشحات هي عبارة عن شيفرات برمجية بسيطة تعمل على إجراء منطق برمجي بسيط وسريع دون الحاجة لاستخدام التوابع أو الخصائص المحسوبة. أما المخاليط فوجدنا أنَّها تقنية مفيدة للتخلص من تكرار الشيفرة البرمجية بين المكونات المختلفة، حيث يُوضع القسم البرمجي المشترك بين مكونين أو أكثر ضمن ملف منفصل دعونه بالمخلوط، وتعمل المكونات المختلفة بعد ذلك على استخدام هذا المخلوط وتتجنب تكرار الشيفرة البرمجية.

# 10. استخدام Vue.js للاتصال بالإنترنت

سنتعلم في هذا الفصل:

- إنشاء قاعدة بيانات على Google Firebase
- تثبيت مكتبة الاتصال بالإنترنت vue-resource
- بناء تطبيق باستخدام Vue.js للقراءة والإضافة من وإلى قاعدة البيانات
- إضافة ميزة تعديل البيانات للتطبيق السابق

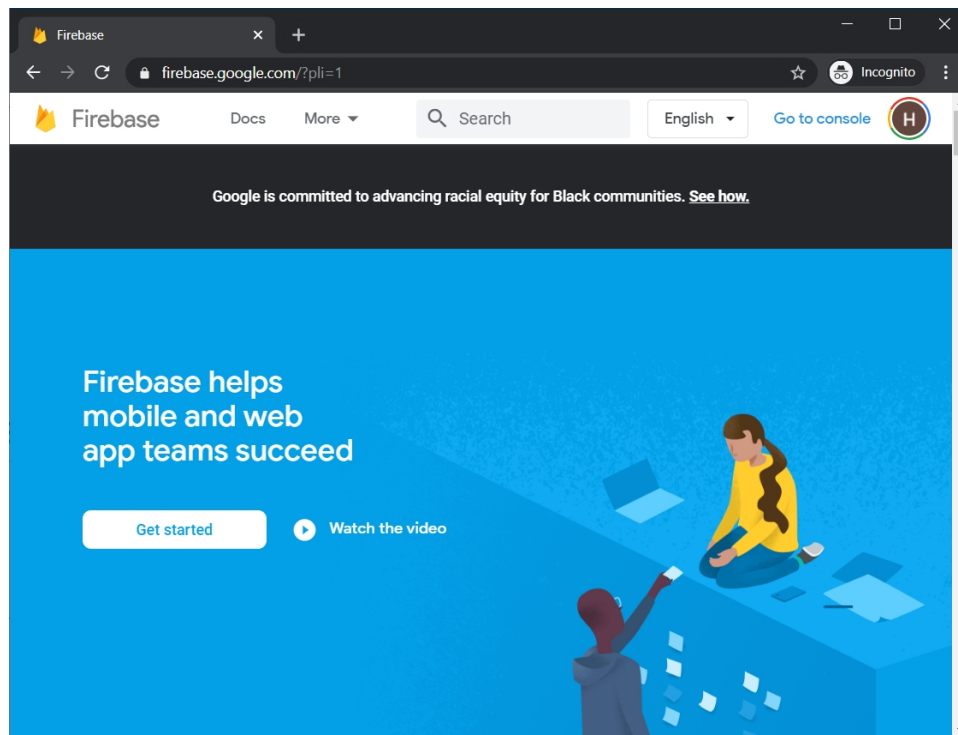
سنتعلم في هذا الفصل كيفية الاتصال بخواديم بعيدة باستخدام مكتبة مخصصة لـ Vue.js لهذا الغرض. وبما أننا نهتم بتبسيط المعلومة من خلال التركيز على مفهوم جديد محدّد، فلن ندخل في مجال بناء تطبيق خلفية كامل (Backend Application) مخصّص لكي يتعامل معه تطبيق Vue.js، ولكن سنعتمد على إنشاء تطبيق بسيط مجاني على Google Firebase، وهو عبارة عن خدمة قاعدة بيانات سيتواصل معها تطبيقنا لتوضيح الفكرة المطلوبة.

## 10.1 إنشاء قاعدة بيانات على Google Firebase

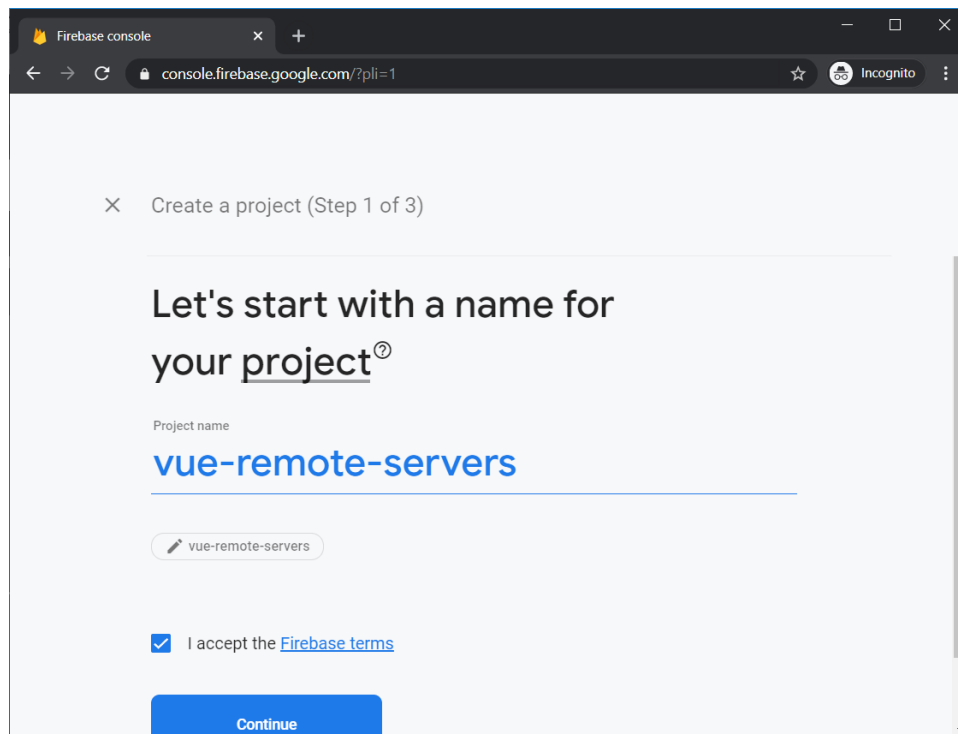
لن نتوسع في هذا الفصل في الشرح حول Google Firebase ومزاياها، إذ يمكنك الاطلاع حول المزيد عنها من خلال الوثائق الخاصة بها والتي سأزودك ببعض منها بعد قليل. يكفيك الآن أن تعلم أن Google Firebase هي منصة تملكها Google وظيفتها مساعدة المطورين على بناء وتحسين وتنمية التطبيقات التي يعملون عليها، فهي عبارة عن خدمة خلفية (Backend Service) يمكنك من خلالها إنشاء قواعد بيانات وخدمات استيثاق (Authentication Services) وغيرها من المزايا المفيدة لتطبيقاتك المختلفة، سواء كانت تطبيقات لأجهزة الجوال أو تطبيقات ويب أو غيرها.

بالنسبة إلينا، سنستخدم Firebase في هذا الفصل لبناء قاعدة بيانات بسيطة كخدمة تمثّل دعمًا لتطبيق Vue.js بسيط. انتقل إلى [الموقع الرسمي لـ Firebase](#)، ثم من الزاوية اليمنى العليا اختر Sign in لتسجيل الدخول، ستحتاج إلى حساب Google للوصول إلى خدمات Firebase.

بعد تسجيل الدخول بحساب Google ستظهر الصفحة الرئيسية التالية:



انقر Go to console من الزاوية اليمنى العليا للانتقال إلى صفحة الخدمات، ثم انقر زر Add project (أو Create a project) لإضافة مشروع جديد، سيطلب منك بدايةً اسم المشروع بالإضافة إلى الموافقة على الاتفاقية كما في الشكل التالي:

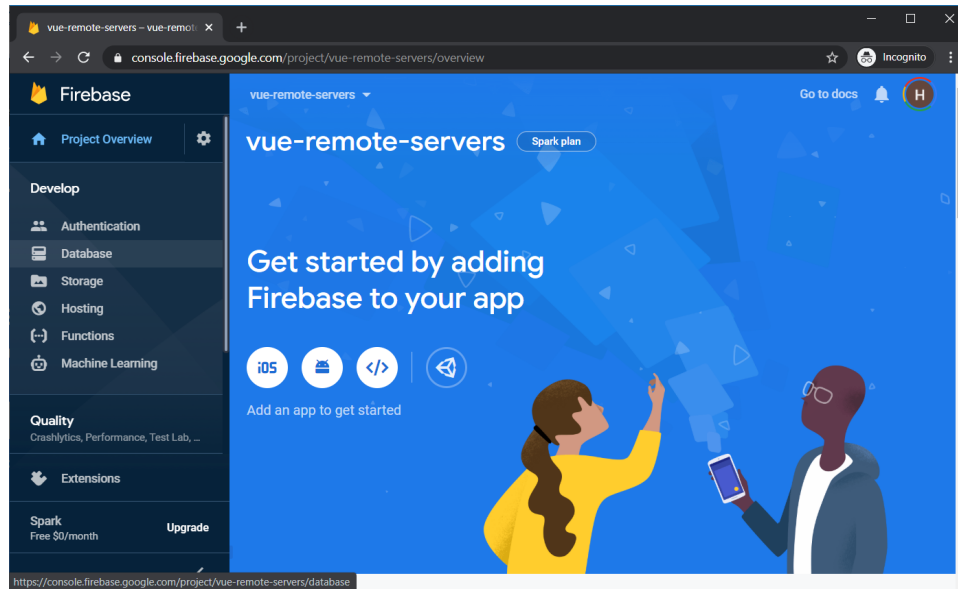


لقد اخترت الاسم `vue-remote-servers` ستضطر بالطبع إلى استخدام اسم آخر لمشروعك لأن الاسم الحالي أصبح محجوزًا. بعد ذلك انقر زر Continue للانتقال إلى المرحلة الثانية التي سيخبرك فيها بتفعيل Google Analytics من أجل هذا المشروع. انقر الزر Continue مجدداً للانتقال إلى المرحلة الأخيرة قبل إنشاء المشروع.

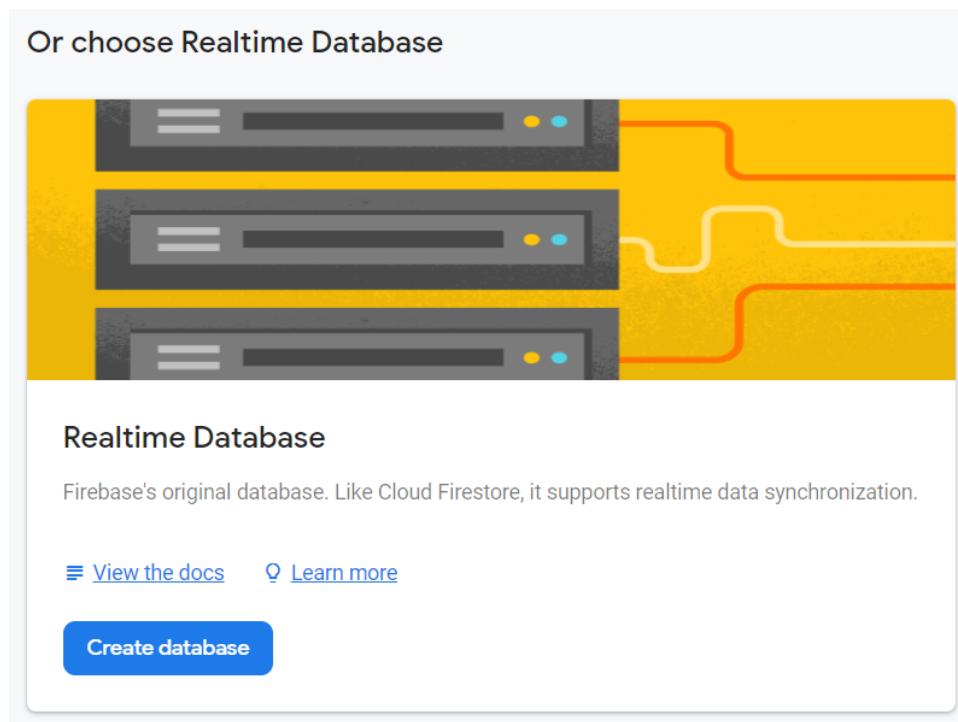
في حال كنت قد اخترت تفعيل Google Analytics في المرحلة الثانية، فسيطلب منك في المرحلة الأخيرة تحديد الحساب الذي ترغب باستخدامه مع Google Analytics (سيوفر لك حساب افتراضي اسمه Default Account For Firebase أو سيسمح لك بإنشاء حساب جديد إن أحببت)، بعد تحديد الحساب، سيظهر زر Create project في الأسفل، انقره لإنشاء المشروع. أمّا إذا لم تفعل Google Analytics من المرحلة الثانية، فسيؤدي ذلك إلى إنشاء المشروع المطلوب فورًا.

إذا كنت تزور Firebase للمرة الأولى، فسيطلب منك في المرحلة الأخيرة تحديد المنطقة (الدولة) الخاصة بـ Google Analytics بالإضافة إلى الموافقة على بنود الاستخدام.

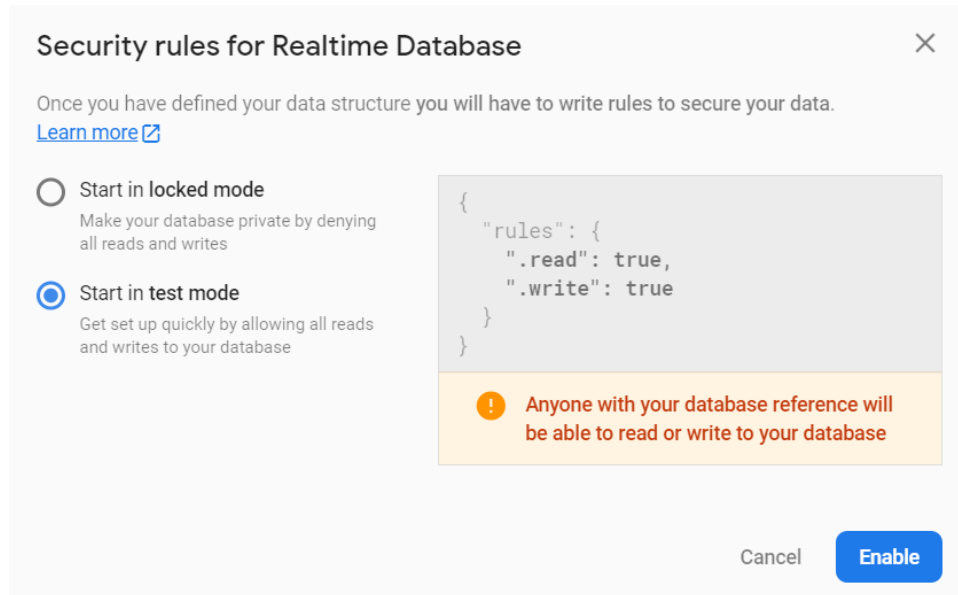
بعد إنشاء المشروع سينتقل المتصفح إلى الصفحة الخاصة به. من القائمة اليسرى انقر الزر Develop ثم انقر Database لأننا لن نهتم الآن سوى بقاعدة البيانات.



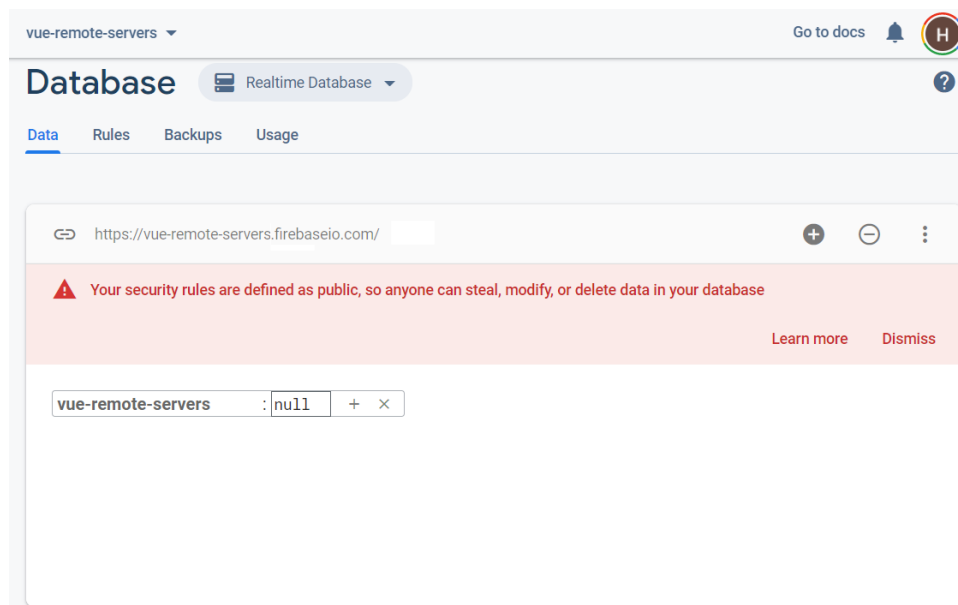
بعد النقر على قاعدة البيانات ستحصل على واجهة تسمح لك ببناء قاعدة بيانات جديدة. استخدم شريط التمرير العمودي لكي تنتقل إلى أسفل الصفحة قليلاً حتى تصل إلى القسم الخاص بإنشاء قاعدة بيانات في الزمن الحقيقي (Realtime Database) كما في الشكل التالي:



انقر زر Create database لإنشاء قاعدة البيانات المطلوبة. ستظهر نافذة صغيرة تعرض فيما إذا كنت تريد أن تجعل قاعدة البيانات مفتوحة للجميع بهدف اختبارها، أم مقفلة تحتاج إلى تسجيل دخول. سنختار أن تكون مفتوحة لجعل الأمر أسهل. اختر Start in test mode:



لاحظ التحذير الذي سيظهر باللون الأحمر والذي يفيد بأن أي شخص سيصبح قادرًا على تعديل قاعدة البيانات أو القراءة منها. بعد النقر على زر Enable ستحصل على شكل شبيه بما يلي:



أصبحت قاعدة البيانات جاهزة، لاحظ من جديد كيف يظهر تحذير باللون الأحمر يخبرك أن قاعدة البيانات مفتوحة ويمكن لأي شخص الوصول إليها. لاحظ أيضًا الرابط الذي يظهر مباشرة فوق التحذير:

`https://vue-remote-servers.firebaseio.com/`

هذا الرابط هو عنوان نقطة الاتصال لخدمة قاعدة البيانات (Database Service Endpoint) التي أنشأناها تـوًا. سنحتاج إلى هذا الرابط لاحقًا في هذا الفصل.



قد تتساءل أيضًا أين الجداول ضمن قاعدة البيانات هذه؟ الحقيقة أن قواعد البيانات ضمن Firebase عبارة عن قواعد بيانات NoSQL لمعرفة المزيد حول قواعد البيانات هذه يمكنك زيارة هذه الصفحة.

يمكنك الحصول على المزيد من المعلومات حول Firebase من خلال هذا الرابط الذي يحتوي على التوثيق الرسمي له ولكن باللغة الإنجليزية.

تحدث في بعض الأحيان تحديثات مستمرة على الواجهات الخاصة بـ Google Firebase، لذلك فقد يختلف الشرح الموجود هنا قليلًا عما تشاهده في الموقع. مع أنني أرجو ألا يحدث ذلك، منعا لأي التباس قد يعتربك عزيزي القارئ.

## 10.2 تثبيت مكتبة الاتصال بالإنترنت vue-resource

يمكن بالطبع استخدام أي طريقة للاتصال بالإنترنت من خلال Vue.js، ولكنني سأختار في هذا الفصل أسلوب مخصص لـ Vue.js. سنستخدم مكتبة اسمها vue-resource يمكن من خلالها الاتصال بالخواديم بشكل سهل وبسيط وبشكل منسجم تقريبًا مع المكتبات الشهيرة المستخدمة لهذا الغرض. يمكنك الوصول إلى المستودع الخاص بهذه المكتبة على Github من خلال هذا الرابط. في الحقيقة، أنصحك بزيارة مستودع هذه المكتبة على Github بعد أن تنتهي من هذا الفصل، لتطلع عليها بشكل جيد وتتعرف على جميع الإمكانيات التي تقدّمها والتي تسهل عملك كمبرمج. في هذا الفصل، سنتحدث عن جزء محدود من هذه المزايا.

لتثبيت vue-resource افتح موجّه الأوامر ونفذ الأمر التالي:

```
npm install vue-resource
```

بعد الانتهاء من التثبيت يمكن الآن إضافة هذه المكتبة إلى تطبيق Vue.js وذلك بإضافة السطر التالي إلى قسم المكتبات المستوردة ضمن الملف main.js:

```
import VueResource from 'vue-resource';
```

واستخدام التعليمة التالية ضمن الملف main.js أيضًا:

```
Vue.use(VueResource);
```

أما كيفية الاستخدام الفعلي لهذه المكتبة فستتعرف عليه في الفقرة التالية.

## 10.3 بناء تطبيق للقراءة والإضافة من وإلى قاعدة البيانات

كما جرت العادة، سنبنّي تطبيق عملي جديد باستخدام Vue CLI سنتعلّم من خلاله كيفية الاتصال بقاعدة البيانات التي أنشأناها في الفقرة الأولى من هذا الفصل على Google Firebase.

أنشئ مشروع جديد اسمه vue-remote-servers عن طريق تنفيذ الأمر التالي ضمن موجه الأوامر:

```
vue create vue-remote-servers
```

بعد إنشاء المشروع، افتح المجلّد الخاص به عن طريق Visual Studio Code. احذف الآن الملف

HelloWorld.vue ثم احرص على أن تكون محتويات الملف main.js مماثلة لما يلي:

```
import Vue from 'vue'
import VueResource from 'vue-resource';
import App from './App.vue'
import "bootstrap/dist/css/bootstrap.min.css";

Vue.config.productionTip = false

Vue.use(VueResource);

new Vue({
  render: h => h(App),
}).$mount('#app')
```

واحرص أيضًا على أن تكون محتويات الملف App.vue مماثلة لما يلي:

```
<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-
offset-3">
        <h2>Vue.js Remote Server</h2>
        <div class="form-group">
          <label>Username</label>
          <input type="text" class="form-control" v-
model="user.username" />
        </div>
        <div class="form-group">
          <label>First name</label>
          <input type="text" class="form-control" v-
model="user.firstname" />
        </div>
        <div class="form-group">
          <label>Last name</label>
          <input type="text" class="form-control" v-
model="user.lastname" />
        </div>
      </div>
    </div>
  </div>
</template>
```

```

        </div>
        <button class="btn btn-primary"
@click="postData()">Add</button>
        <br />
        <br />
        <button class="btn btn-primary"
@click="getData()">Retrieve</button>
        <br />
        <br />
        <ul class="list-group">
          <li
            class="list-group-item"
            v-for="usr in users"
            v-bind:key="usr.username"
          >{{usr.username}} - {{usr.firstname}} {{usr.lastname}}</li>
        </ul>
      </div>
    </div>
  </div>
</template>

<script>
export default {
  name: "App",
  data() {
    return {
      user: {
        username: "",
        firstname: "",
        lastname: "",
      },
      users: [],
    };
  },
  methods: {
    postData: function () {
      this.$http
        .post("https://vue-remote-servers.firebaseio.com/users.json",
this.user)
        .then(
          (response) => {
            console.log(response);

            this.user.username = "";
            this.user.firstname = "";
            this.user.lastname = "";
          },
          (error) => {
            console.log(error);
          }
        );
    },
  },

```

```

    getData: function () {
      this.$http
        .get("https://vue-remote-servers.firebaseio.com/users.json")
        .then((response) => {
          return response.json();
        })
        .then((data) => {
          const tmpArray = [];

          for (let key in data) {
            tmpArray.push(data[key]);
          }

          this.users = tmpArray;
        });
    },
  },
};
</script>

<style>

</style>

```

ستعطي الشيفرة السابقة الواجهة التالية:

## VueJS Remote Server

Username

First name

Last name

Add

Retrieve

الفكرة من هذا التطبيق هي في إمكانية الإضافة المتعددة لمستخدمين افتراضيين. حيث تتكون بيانات كل مستخدم من: اسم المستخدم (Username) والاسم (First name) والكنية (Last name). سترسل

بيانات كل مستخدم مفترض إلى قاعدة البيانات على Firebase. بعد ذلك يمكن استرداد بيانات المستخدمين المضافة مسبقًا إلى قاعدة البيانات هذه، من خلال نقر الزر Retrieve.

انتبه إلى أننا سنستخدم في هذا التطبيق تنسيقات Bootstrap على افتراض أنك قد ثبتته مسبقًا. وهذا ما يفسّر وجود السطر التالي ضمن ملف main.js:

```
import "bootstrap/dist/css/bootstrap.min.css";
```

إذا أردت أن تتذكر كيفية استخدام تنسيقات Bootstrap ضمن Vue.js يمكنك مراجعة الفقرة: "استخدام إطار العمل Bootstrap" ضمن الفصل: "التعامل مع دخل المستخدم عن طريق نماذج الإدخال".

بالنسبة لآلية عمل التطبيق، لاحظ معي بدايةً أن شيفرة HTML الموجودة ضمن القسم <template> هي بسيطة وواضحة. بالنسبة لحقول البيانات المستخدمة، فهي معرفة ضمن القسم <script> حيث عرّفت جميع الحقول التي تعود للمستخدم المفترض ضمن كائن اسمه user موجود ضمن القسم data ضمن كائن Vue.js. يحتوي هذا الكائن على الحقول التالية: username و firstname و lastname كما هو واضح. كما يحتوي القسم data أيضًا على مصفوفة اسمها users سنخزن ضمنها بيانات المستخدمين التي سنحصل عليها من قاعدة بيانات Firebase عند نقر زر Retrieve كما سنوضح ذلك بعد قليل.

أما بالنسبة للقسم methods فيحتوي على تابعين فقط: postData و getData وهما معالجين لحدثي النقر على الزرين Add و Retrieve على الترتيب.

بالنسبة للتابع postData تأمل معي الشيفرة البرمجية الموجودة ضمنه:

```
this.$http
  .post("https://vue-remote-servers.firebaseio.com/users.json",
  this.user)
  .then(
    (response) => {
      console.log(response);

      this.user.username = "";
      this.user.firstname = "";
      this.user.lastname = "";
    },
    (error) => {
      console.log(error);
    }
  );
```

استخدمت الكائن `$http` من الكائن `this`. في الحقيقة أن هذا الكائن مُتاح فقط من خلال المكتبة `vue-resource` التي أضفناها في هذا الفصل. يحتوي الكائن `$http` على عدة توابع مفيدة في استقبال وإرسال البيانات عبر الإنترنت من وإلى خواديم أو خدمات (Services) كما هي خدمة `Firebase`. بما أننا نريد إضافة بيانات جديدة، فسنستخدم التابع `post` كما هو واضح. يقبل هذا التابع وسيطين:

الوسيط الأول هو عنوان نقطة الاتصال للخدمة المراد التواصل معها، وهي في حالتنا هذه عنوان خدمة قاعدة البيانات على `Firebase` التي أنشأناها في هذا الفصل:

```
https://vue-remote-servers.firebaseio.com/users.json
```

الرابط السابق حصلت عليه من الصفحة الخاصة بقاعدة البيانات `vue-remote-servers` إذا كنت تذكر من الفقرة الأولى من هذا الفصل، ولكن أضفت عليه اسم المصدر `users.json`. بما أنني أريد إضافة مستخدمين افتراضيين إلى قاعدة البيانات فمن المنطقي إضافة بيانات هؤلاء المستخدمين إلى جدول (إن صح التعبير) اسمه `users`. في الحقيقة في قواعد البيانات من النمط `NoSQL` نسمي مثل هذه الجداول بالمجموعات (Collections). إذا فالاسم `users` هو اسم المجموعة التي سنخزن ضمنها بيانات المستخدمين الافتراضيين، أما الامتداد `.json` فهو إلزامي.

أما الوسيط الثاني فهو عبارة عن البيانات المراد إرسالها. وقد أرسلت الكائن `user` جملة واحدة، لأنه يحتوي على البيانات الفرعية المطلوبة والتي هي مربوطة بدورها عن طريق `v-model` بعناصر `HTML` الموافقة.

يُرجع التابع `post` وعد (Promise) لذلك فإننا نَتبع استدعاء التابع `post` بنقطة مباشرة وبعدها نكتب التابع `then` كما هو مألوف تمامًا عند العمل مع مكتبات `JavaScript` الشهيرة الأخرى. يقبل التابع `then` وسيطين أيضًا: الوسيط الأول هو تابع يحتوي على شيفرة برمجية تُعالج الرد الذي حصلت عليه مكتبة `vue-resource` بعد اتصالها مع الخادوم البعيد. انظر إلى الشيفرة التالية:

```
(response) => {
  console.log(response);

  this.user.username = "";
  this.user.firstname = "";
  this.user.lastname = "";
}
```

مررنا التابع السابق كتابع سهم (Arrow Function) يقبل وسيطًا اسمه `response` وهو يمثل الرد الذي حصلت عليه المكتبة من الخادوم. ضمن هذا التابع وفي السطر الأول منه نلاحظ وجود

التعليمة `console.log(response)` لعرض محتوى البيانات الواردة من الخادوم ضمن الطرفية (Console) الخاصة بأدوات المطور ضمن متصفح الويب. يمكنك إزالة هذا السطر إن أحببت. أما التعليمات الثلاث التالية فهدفها تفريغ حقول بيانات المستخدم للإشارة إلى نجاح العملية.

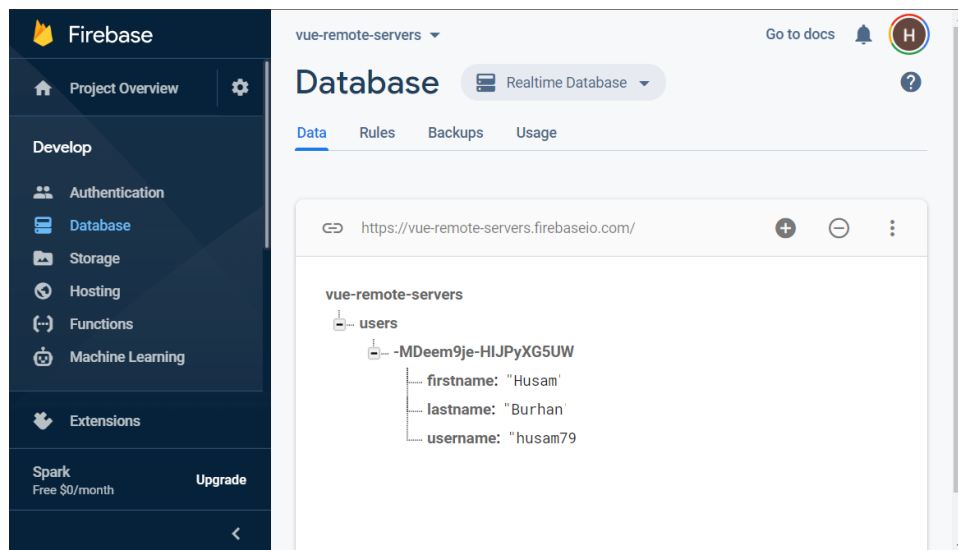
أما الوسيط الثاني للتابع `then` فهو أيضًا تابع آخر، ولكنه يحتوي على الشيفرة البرمجية التي تعالج حالة وجود خطأ ما في عملية الإرسال.

عند هذه النقطة بالتحديد، أفضل تشغيل التطبيق لنرى ماذا سيحدث عند إضافة مستخدم جديد. شغل التطبيق عن طريق تنفيذ الأمر التالي ضمن موجه الأوامر وضمن المجلد `vue-remote-servers` الذي يحوي ملفات التطبيق:

```
npm run serve
```

رُ الآن الصفحة <http://localhost:8080> لتحصل على واجهة التطبيق. أدخل قيم مناسبة لاسم المستخدم و الاسم والكنية، ثم انقر الزر Add. إذا اختفت القيم التي أدخلتها تَوَّ هذا دليل على نجاح العملية! انتقل الآن إلى الصفحة الخاصة بقاعدة البيانات `vue-remote-servers` ([عليك زيارة الموقع](#)) ثم نقر زر Go to console وبعد ذلك اختيار المشروع `vue-remote-servers` ثم نقر زر Database من القائمة اليسرى، وأخيرًا اختيار قاعدة البيانات الحقيقية (Realtime Database) التي تحمل نفس الاسم).

بحسب البيانات التي أدخلتها أنا، حصلت على الشكل التالي:



لاحظ معي اسم قاعدة البيانات `vue-remote-servers` يظهر في أعلى الشجرة، ثم يظهر اسم المجموعة `users` في العقدة التي تليها. يتفرع عن المجموعة `users` التي هي بمثابة جدول كما اتفقنا، عقدة تالية تحمل

رموزًا مبهمًا، يتفرّع عنها من جديد البيانات التي أرسلتها إلى قاعدة البيانات وهي: Husam للاسم و Burhan للكنية و husam79 لاسم المستخدم.

بالنسبة للرموز المبهمة التي تظهر فهي بمثابة مفتاح رئيسي (Primary Key) أو معرف (ID) للبيانات التي تقع ضمنها. لأنه من البديهي إرسال العديد من بيانات المستخدمين، فسيعمل Firebase على تخصيص شيفرة فريدة تحوي مثل هذه الرموز المبهمة لكل مستخدم مفترض تتم إضافته إلى قاعدة البيانات. يمكنك إن أحببت إضافة مستخدم آخر لترى كيف يحدث ذلك.

كل الكلام السابق كان للتابع postData. أما بالنسبة للتابع getData فهي الشيفرة البرمجية الموجودة ضمنه:

```

this.$http
  .get("https://vue-remote-servers.firebaseio.com/users.json")
  .then((response) => {
    return response.json();
  })
  .then((data) => {
    const tmpArray = [];

    for (let key in data) {
      tmpArray.push(data[key]);
    }

    this.users = tmpArray;
  });

```

سنستخدم هذه المرة التابع get من الكائن \$http وذلك للحصول على البيانات المخزنة ضمن قاعدة البيانات. لا يحتاج هذا التابع كما هو واضح إلا لوسيط واحد هو نفسه الوسيط الأول للتابع post الذي تحدثنا عنه قبل قليل.

يُرجع التابع get أيضًا وعد (Promise)، لذلك وبنفس النقاش السابق فإننا نُنْبَع استدعاء التابع get بنقطة مباشرة وبعدها نكتب التابع then. اكتفيت هنا بكتابة وسيط واحد للتابع then -وهذا جائز تمامًا- وهو عبارة عن تابع سهم كما هو واضح يحتوي على تعليمة برمجية واحدة فقط:

```

return response.json();

```

وظيفة هذه التعليمة هي إرجاع تمثيل JSON للرد الذي حصلت عليه المكتبة بعد استدعاء التابع get. وذلك من خلال استدعاء التابع json من الكائن response. وهنا ينبغي التنويه إلى أن التابع json يُرجع هو أيضًا وعد (Promise). لذلك فإن التعليمة السابقة تُرجع هذا الوعد مما يسمح لنا بوضع نقطة مباشرة بعد



استدعاء التابع then ثم استدعاء تابع then جديد يقبل وسيط وهو بالطبع عبارة عن تابع سهم جديد يقبل وسيطًا واحدًا أيضًا أسميته data يحتوي على البيانات الفعلية التي حصلنا عليها من قاعدة البيانات على Firebase.

ما تبقى ضمن تابع السهم الموجود ضمن تابع then الأخير عبارة عن شيفرة برمجية وظيفتها استخلاص البيانات الخام الواردة من المكتبة vue-resource التي تتواصل مع قاعدة البيانات، وتخزينها بشكل مناسب ضمن المصفوفة المؤقتة tmpArray. وبعد انتهاء عملية الاستخلاص، تُسند هذه المصفوفة إلى الحقل users وهو مصفوفة بالطبع، لتعرض البيانات بالشكل المرغوب على المستخدم. انظر هذه الشيفرة:

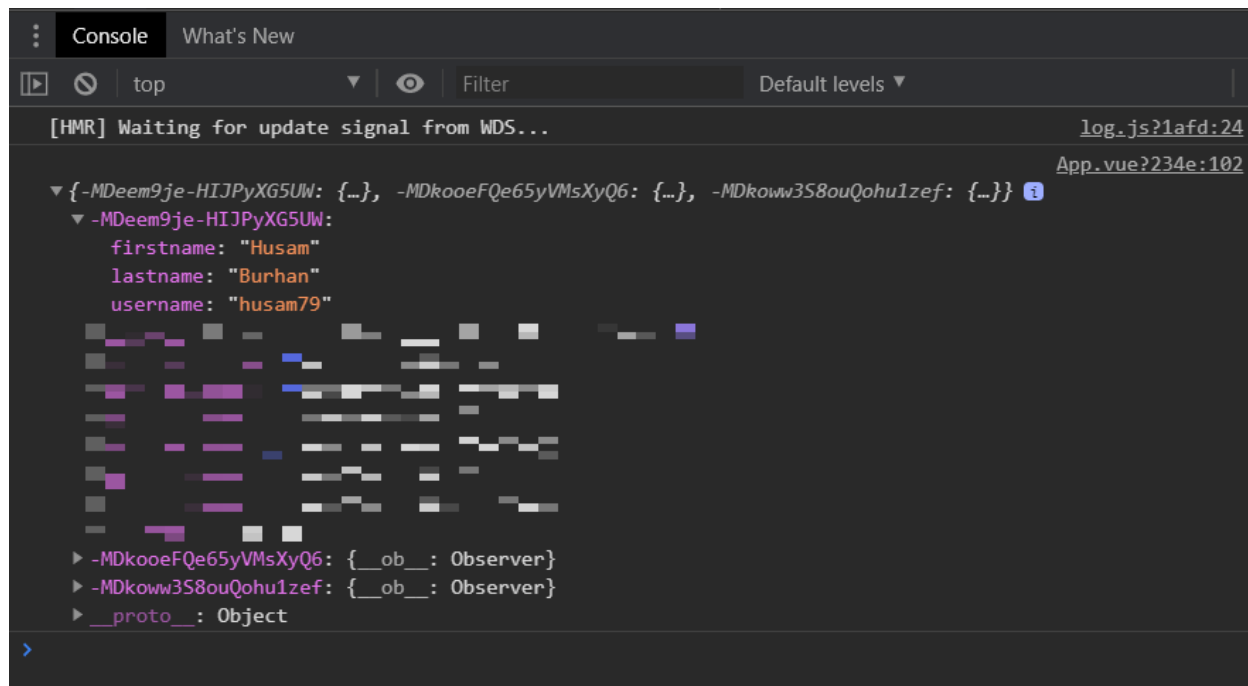
```
const tmpArray = [];

for (let key in data) {
  tmpArray.push(data[key]);
}

this.users = tmpArray;
```

إن سبب وجود حلقة for بالشكل السابق، يعود إلى شكل البيانات الواردة من المكتبة vue-resource والتي هي عبارة عن قاموس (Dictionary)، يكون المفتاح (Key) فيه عبارة عن المفتاح الرئيسي (تلك الرموز الغريبة) لأحد المستخدمين، أما القيمة (Value) لهذا المفتاح فهي عبارة عن كائن يتضمن بيانات المستخدم (في حالتنا هذه تكون هذه البيانات عبارة عن اسم المستخدم والاسم والكنية كما نعلم).

انظر معي إلى البيانات الخام الواردة من المكتبة vue-resource، والتي حصلت عليها بإضافة تعليمة الكتابة إلى الطرفية (Console) عند ورود البيانات من المكتبة (في الحقيقة تعمدت إخفاء بعض التوابع الأخرى الموجودة ضمن الكائن للتركيز على ما يهمنا هنا):



الشكل السابق هو نتيجة إضافة مستخدمين افتراضيين آخرين إلى قاعدة البيانات، وبعد نقر زر Retrieve بالطبع. انظر إلى الشكل التالي لترى كيف ستبدو نفس البيانات ولكن على صفحة الويب:

## VueJS Remote Server

Username

First name

Last name

Add

Retrieve

husam79 - Husam Burhan

jamil2020 - Jamil Bailony

hsoub - Hsoub Academy

## 10.4 إضافة ميزة تعديل البيانات للتطبيق السابق

من الواضح أن تطبيقنا السابق يسمح بإضافة مستخدمين افتراضيين جدد، كما يسمح بقراءة بيانات هؤلاء المستخدمين وعرضهم على الصفحة. ولكن من المفيد أن يسمح التطبيق أيضًا بتعديل بيانات مستخدم افتراضي موجود مسبقًا. يمكن ذلك ببساطة من خلال استخدام التابع put من الكائن \$http حيث يسمح هذا التابع بتعديل بيانات مستخدم موجود مسبقًا ضمن قاعدة البيانات بمجرد أن نعرف المفتاح الرئيسي (تلك الرموز الغريبة) للمستخدم ضمن قاعدة البيانات.

سأجري في الحقيقة تعديلًا كبيرًا من الناحية الشكلية على التطبيق السابق لكي يسمح بإنجاز هذه الميزة، ورغم التعديلات الكثيرة التي ستجري على الملف App.vue، إلا أنها تُعتبر في حقيقة الأمر بسيطة وواضحة خصوصًا بعد أن وصلنا إلى هذه المرحلة في العمل مع Vue.js.

إليك الشيفرة البرمجية الجديدة للملف App.vue:

```
<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-
offset-3">
        <h2>Vue.js Remote Server</h2>
        <div class="form-group">
          <label>Username</label>
          <input type="text" class="form-control" v-
model="user.username" />
        </div>
        <div class="form-group">
          <label>First name</label>
          <input type="text" class="form-control" v-
model="user.firstname" />
        </div>
        <div class="form-group">
          <label>Last name</label>
          <input type="text" class="form-control" v-
model="user.lastname" />
        </div>
      </div>
      <div class="row">
        <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-md-6 col-md-
offset-3">
          <button class="btn btn-primary"
@click="postOrPutData()">{{actionButtonText}}</button>
          <button class="btn btn-primary float-right"
@click="reset()">Reset</button>
```

```

        </div>
      </div>
      <hr>
      <div class="row">
        <div class="col-2">
          <button class="btn btn-primary btn-dark"
@click="getData()">Retrieve</button>
        </div>
      </div>
      <div class="row" v-for="usr in users" v-bind:key="usr.username">
        <div class="col-2">{{usr.username}}</div>
        <div class="col-4">{{usr.firstname}} {{usr.lastname}}</div>
        <div class="col-1">
          <button class="btn btn-link"
@click="prepareToSave(usr.id)">Edit</button>
        </div>
      </div>
    </div>
  </template>

  <script>
  export default {
    name: "App",
    data() {
      return {
        user: {
          username: "",
          firstname: "",
          lastname: "",
        },
        users: [],
        currentUserIdToSave: "",
        actionButtonTitle: "Add",
      };
    },
    methods: {
      postOrPutData: function () {
        if (this.actionButtonTitle === "Add") {
          this.$http
            .post(
              "https://vue-remote-servers.firebaseio.com/users.json",
              this.user
            )
            .then(
              (response) => {
                console.log(response);

                this.user.username = "";
                this.user.firstname = "";
                this.user.lastname = "";
              },
              (error) => {

```

```

        console.log(error);
    }
    );
} else {
    this.$http
        .put(
            "https://vue-remote-servers.firebaseio.com/users/" +
            this.currentUserIdToSave +
            ".json",
            this.user
        )
        .then(
            (response) => {
                console.log(response);
            },
            (error) => {
                console.log(error);
            }
        );
}
},
getData: function () {
    this.$http
        .get("https://vue-remote-servers.firebaseio.com/users.json")
        .then((response) => {
            return response.json();
        })
        .then((data) => {
            const tmpArray = [];

            for (let key in data) {
                let withId = data[key];
                withId.id = key;
                tmpArray.push(data[key]);
            }

            this.users = tmpArray;
        });
},
prepareToSave: function (id) {
    for (var i = 0; i < this.users.length; i++) {
        if (this.users[i].id === id) {
            this.user.username = this.users[i].username;
            this.user.firstname = this.users[i].firstname;
            this.user.lastname = this.users[i].lastname;
            this.currentUserIdToSave = this.users[i].id;

            this.actionButtonTitle = "Save";

            break;
        }
    }
}
}

```

```

    },
    reset: function () {
      this.actionButtonTitle = "Add";
      this.currentUserIdToSave = "";

      this.user.username = "";
      this.user.firstname = "";
      this.user.lastname = "";
    },
  },
};
</script>

<style>
.row {
  margin-top: 8px;
}
</style>

```

سأوضح التعديلات الرئيسية التي طرأت على الشيفرة البرمجية. في البداية أجريت تعديلات تنسيقية على القسم `<template>` للسماح بإضافة زر جديد اسمه `Reset`، بالإضافة إلى أنني غيرت العنصر المسؤول عن عرض بيانات المستخدمين من قاعدة البيانات (كان عبارة عن قائمة غير مرتبة `ul`)، وذلك لكي أستطيع إضافة زر تعديل `Edit` بجوار كل مستخدم للسماح بتعديل بياناته في حال الرغبة بذلك.

أضفت أيضًا حقلين جديدين للقسم `data`، الأول هو `currentUserIdToSave` ووظيفته الاحتفاظ بالمفتاح الرئيسي للمستخدم الحالي الذي نرغب بتعديل بياناته، أما الحقل الثاني فهو `actionButtonTitle` والذي جعلت قيمته الحالية تظهر مباشرة على الزر `Add` الذي سيتغير النص الظاهر عليه بحسب السياق. أي عندما نرغب بإضافة مستخدم جديد سيظهر النص `Add`، أما عندما نرغب بتعديل بيانات مستخدم موجود مسبقًا سيظهر النص `Save`.

بمعنى آخر، سيعمل التطبيق على تنفيذ مجموعة مختلفة من التعليمات البرمجية بحسب النص الذي يظهر حاليًا على هذا الزر. لذلك فمن المنطقي تغيير اسم التابع الذي سيُنَفَّذ عند النقر على هذا الزر، فقد غيرت اسم التابع من `postData` إلى `postOrPutData` للإشارة إلى وظيفته الجديدة المتمثلة في الإضافة الجديدة أو التعديل على بيانات موجودة مسبقًا. انظر إلى الشيفرة البرمجية الموجودة ضمن هذا التابع:

```

if (this.actionButtonTitle === "Add") {
  this.$http
    .post(
      "https://vue-remote-servers.firebaseio.com/users.json",

```

```

        this.user
      )
      .then(
        (response) => {
          console.log(response);

          this.user.username = "";
          this.user.firstname = "";
          this.user.lastname = "";
        },
        (error) => {
          console.log(error);
        }
      );
    } else {
      this.$http
        .put(
          "https://vue-remote-servers.firebaseio.com/users/" +
            this.currentUserIdToSave +
            ".json",
          this.user
        )
        .then(
          (response) => {
            console.log(response);
          },
          (error) => {
            console.log(error);
          }
        );
    }
  }
}

```

لاحظ عبارة `if` الموجودة في بداية التابع، وانتبه إلى الشرط الموجود ضمنها. يختبر هذا الشرط فيما إذا كانت القيمة الحالية للحقل `actionButtonTitle` تساوي القيمة `Add`. فإذا تحقق هذا الشرط، فهذا يعني أننا نريد إضافة مستخدم جديد، فثنقذ الكتلة البرمجية المتعلقة بتحقيق ذلك الشرط، وهي نفس الكتلة البرمجية التي كانت موجودة ضمن التابع `postData` قبل التعديل. أمّا إذا لم يتحقق الشرط، فهذا يعني بالتأكيد أننا في السياق الذي يسمح بتعديل بيانات موجودة مسبقاً، لذلك ثنقذ الكتلة البرمجية الموافقة التي تستخدم في هذه المرة التابع `put` كما هو واضح. يتم التحكم في قيمة الحقل `actionButtonTitle` ضمن التابعين الجديدين `prepareToSave` و `reset` كما سنرى ذلك بعد قليل. لنركّز الآن قليلاً على التابع `put`:

```

this.$http
  .put(
    "https://vue-remote-servers.firebaseio.com/users/" +
      this.currentUserIdToSave +

```

```

      ".json",
      this.user
    )
  }
}

```

يقبل هذا التابع وسيطين. الوسيط الأول هو عنوان المصدر الذي سنعمل على تعديل بياناته. لاحظ معي كيف وضعت اسم الجدول (المجموعة) `users` يليه قيمة المفتاح الرئيسي الذي حصلت عليه من قيمة الحقل `currentUserIdToSave` ثم وضعت الامتداد الإلزامي `..json`. أمّا الوسيط الثاني فهو بكل بساطة الكائن `user` الذي من المفترض الآن أن يحمل البيانات المعدلة لهذا المستخدم. من الممكن أن نجري تعديل على جزء من البيانات أو على جميع البيانات أو أن لا نجري أية تعديلات. في كل حالة من الحالات السابقة سيتم استبدال بالقيم الحالية لبيانات الكائن `user` البيانات القديمة الموجودة ضمن قاعدة البيانات والتي لها نفس قيمة المفتاح الرئيسي.

كما أوضحت قبل قليل، فقد أضفت تابعين جديدين هما: `prepareToSave` و `reset`، كما أجريت تعديلاً بسيطاً للتابع `getData`. سأبدأ من التعديل البسيط الذي أجرّيته على التابع `getData`. أضفت في الحقيقة حقلاً جديداً إلى كل كائن يمثل مستخدم مفترض تم جلبه من قاعدة بيانات. انظر معي إلى حلقة `for` التي حدث فيها التعديل:

```

for (let key in data) {
  let withId = data[key];
  withId.id = key;
  tmpArray.push(data[key]);
}

```

أنشأت متغير جديد اسمه `withId` تنحصر وظيفته في إضافة الحقل `id` إلى البيانات التي تأتي أصلاً من قاعدة البيانات وذلك بهدف الاحتفاظ بالمفتاح الرئيسي لكل مستخدم مع بياناته الأساسية. سنرى سبب هذا التعديل بعد لحظات.

بالنسبة للشفيرة البرمجية لكل من التابعين `reset` و `prepareToSave` فهي بسيطة للغاية. بالنسبة للتابع `prepareToSave` فيُستدعى عند نقر الزر `Edit` بجوار مستخدم ما. يقبل هذا التابع وسيطاً واحداً هو قيمة المفتاح الرئيسي لذلك المستخدم الذي نرغب بتعديل بياناته. بعد ذلك ندخل حلقة `for`، هدفها الوصول إلى الكائن الذي يمثل المستخدم المراد تعديل بياناته. استطعت تحديد هذا الكائن، وذلك بمقارنة قيمة الوسيط `id` (الذي يحمل قيمة المفتاح الرئيسي للمستخدم المراد تعديله) مع قيمة الحقل `id` لكل مستخدم موجود ضمن المصفوفة `users`. لاحظ هنا أنّ الحقل `id` الموجود ضمن المصفوفة `users` قد أضفناه ضمن التابع `getData`.



بعد الوصول إلى الكائن المطلوب ضمن المصفوفة، تعمل الشيفرة على تحديث بيانات الحقل `user` وبالتالي تتم تعبئة عناصر الإدخال على الصفحة ببيانات المستخدم المراد تعديله، ثم تُعدّل قيمة الحقل `currentUserIdToSave` ليحمل قيمة المفتاح الرئيسي لهذا المستخدم المراد تعديله (لاحظ أننا استخدمنا قيمة هذا الحقل ضمن التابع `postOrPutData` بتمريره للتابع `put`)، وأيضًا تُعدّل قيمة الحقل `actionButtonTitle` لتحمل القيمة `Save`، وهكذا نكون قد دخلنا في سياق حفظ البيانات، أي نكون جاهزين لتعديل بيانات المستخدم.

وأخيرًا بالنسبة للتابع `reset` فهو يُعيد الأمور إلى أصلها الأول. أي يعمل على تفريغ حقول الإدخال، ويعمل على إعادة السياق إلى حالة إدخال مستخدم جديد بإسناد القيمة `Add` ضمن الحقل `actionButtonTitle`، كما يعمل على تفريغ الحقل `currentUserIdToSave` لأنه لا يوجد حاليًا أي مستخدم نرغب بتعديل بياناته كما هو واضح.

إذا شغلت التطبيق بعد هذه التعديلات يُفترض أن تحصل على شكل شبيه بما يلي (لاحظ أنني قد نقرت أيضًا الزر `Retrieve`):

## VueJS Remote Server

Username

First name

Last name

Add

Reset

Retrieve

husam79

Husam Burhan

[Edit](#)

jamil2020

Jamil Bailony

[Edit](#)

hsoub

Hsoub Academy

[Edit](#)

جرب نقر زر Edit بجوار أحد المستخدمين، ثم عدّل بياناته، ثم انقر زر Save، وبعدها انقر زر Retrieve من جديد. يجب أن ترى الآن التعديلات التي أجريتها على ذلك المستخدم. انقر زر Reset للعودة إلى سياق إضافة مستخدم جديد.

لا يُعتبر الأسلوب الذي اتبعته في تعديل البيانات أسلوبًا عمليًا في التطبيقات الحقيقية. إذ كان من الواجب الانتقال إلى صفحة منفصلة عند النقر على زر Edit لتعديل بيانات المستخدم بشكل مستقل. لكنني آثرت اتباع هذا الأسلوب غير العملي الآن، لأنّ اتباع الأسلوب العملي يقتضي الدخول في بحث جديد تمامًا وهذا ما سأفعله لاحقًا في فصل منفصل إن شاء الله.

من الممكن أيضًا إضافة ميزة حذف مستخدم موجود مسبقًا باستخدام التابع delete مع الكائن \$http مع تمرير وسيط واحد فقط له. هو نفسه الوسيط الأول للتابع put.

## 10.5 ختام الفصل

لقد تعلّمنا الكثير في هذا الفصل كيفية إنشاء قاعدة بيانات بسيطة على Google Firebase، كما وتعلّمنا كيفية استخدام المكتبة vue-resource لإكساب تطبيقات Vue.js القدرة على الاتصال بالإنترنت، وبنينا أيضًا تطبيق عملي مبسّط يوضّح كيفية التواصل مع قاعدة البيانات على Google Firebase، وبالتالي كيفية إضافة وقراءة وتعديل البيانات الموجودة ضمنها.

يعد هذا الفصل مهمًا بالفعل، فمن خلاله استطعت للمرة الأولى الخروج من "القمقم" والتواصل مع العالم الخارجي. ستحمل ما تبقى من فصول مفاهيم مهمّة أيضًا حول كيفية التعامل مع Vue.js تلك المكتبة القوية والمرنة.

# 11. بناء تطبيقات ذات صفحة واحدة

سنتعلم في هذا الفصل:

- بناء هيكل التطبيق والتعرف على أجزائه الرئيسية.
- تحويل التطبيق الموجود في الفصل السابق إلى تطبيق SPA.

سنتعلم في هذا الفصل كيفية بناء تطبيقات تعتمد على صفحة واحدة فقط (Single Page Applications) أو اختصارًا SPA. توفر Vue.js هذه الإمكانية من خلال مكتبة اسمها vue-router تسمح بتعريف مسارات داخلية ضمن التطبيق بهدف دعم مفهوم SPA. وتطبيقات SPA هي تطبيقات ويب عادية، لكنها تختلف عن التطبيقات الكلاسيكية في أن الانتقال من صفحة إلى أخرى ضمن الموقع لا يحتاج إلى إعادة تحميل كامل الصفحة بما تحويه من ملفات شيفرة نصية وصور وغيرها من أصول الموقع. فالذي يحدث في تطبيقات SPA هو أن جميع الواجهات المفترض وجودها في الموقع ستكون معروفة مسبقًا ومحملة إلى حاسوب المستخدم، فيعمل تطبيق SPA فقط على استبدال واجهة مكان واجهة أخرى دون الحاجة إلى تحميل الصفحة كاملةً من الخادوم.

يرتبط هذا الفصل ارتباطًا وثيقًا بالفصل السابق لأنه سيعمل على تحويل التطبيق المعتمد في الفصل السابق إلى تطبيق يدعم SPA لذلك أنصحك بمراجعة الفصل السابق قبل الاستمرار في هذا الفصل. على أية حال، أرجو أن يكون هذا الفصل ممتعًا ومفيدًا.

## 11.1 بناء هيكل التطبيق والتعرف على أجزائه الرئيسية

بنينا في الفصل السابق تطبيق تجريبي بسيط، هدفه التواصل مع قاعدة بيانات موجودة على Firebase، ولعلك تذكر، أنني قد أشرت في ذلك الفصل إلى أن الأسلوب الذي اتبعناه في بناء ذلك التطبيق لم يكن عمليًا بسبب أننا قد "حشرنا" جميع وظائف التطبيق ضمن واجهة واحدة وهذا أمر غير عملي بالطبع. حان الآن إصلاح ذلك العيب وذلك باستخدام تقنية SPA. سنحوّل جميع وظائف التطبيق السابق إلى تطبيق جديد سنبنيه في هذا الفصل.

لنبدأ بإنشاء هيكل التطبيق الجديد بتنفيذ الأمر التالي ضمن موجه الأوامر:

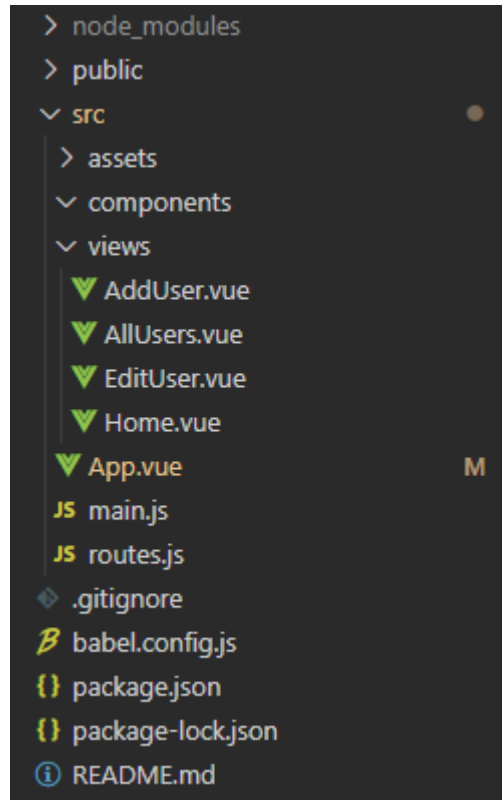
```
vue create vue-spa
```

لكي ننجز مفهوم SPA في تطبيقنا هذا، سنستخدم تقنية التوجيه (Routing) التي عن طريق المكتبة vue-router كما أشرنا قبل قليل. استخدم الأمر التالي لتثبيت المكتبة vue-router:

```
npm install vue-router
```

افتح مجلد التطبيق باستخدام Visual Studio Code ثم احذف الملف HelloWorld.vue. سنحتاج إلى إنشاء مجموعة جديدة من الملفات والمجلدات من أجل هذا التطبيق. انقر الآن بزر الفأرة الأيمن على المجلد src واختر New Folder لإنشاء مجلد جديد سمّه views. سنستخدم هذا المجلد لتخزين الواجهات المختلفة للتطبيق الخاص بنا. أنشئ ضمن المجلد views الذي أنشأناه تَوًّا الملفات التالية: AddUser.vue و EditUser.vue و AllUsers.vue و Home.vue. أنشئ أيضًا الملف routes.js ضمن المجلد src. الملف routes.js سيحتوي على مسارات العناوين الداخلية التي سنستخدمها ضمن التطبيق.

ستحصل بالنتيجة على البنية التالية من الملفات:



## 11.2 تحويل تطبيق إلى تطبيق SPA

لكي نبدأ باستخدام التوجيه، يجب أولاً أن نستورد المكتبة `vue-router` عن طريق عبارة الاستيراد التالية:

```
import VueRouter from "vue-router";
```

سنضع هذه العبارة (وبشكل مختلف عما اعتدنا عليه) ضمن الملف `routes.js` وسأوضح سبب ذلك بعد

قليلاً. انظر الآن إلى محتويات الملف `routes.js`:

```
import Vue from "vue";
import VueRouter from "vue-router";
import Home from "./views/Home";
import AddUser from "./views/AddUser";
import EditUser from "./views/EditUser";
import AllUsers from "./views/AllUsers";
```

```
Vue.use(VueRouter);
```

```
const routes = [
  {
    path: "/",
    name: "Home",
    component: Home
```

```

    },
    {
      path: "/AddUser",
      name: "AddUser",
      component: AddUser
    },
    {
      path: "/EditUser",
      name: "EditUser",
      component: EditUser
    },
    {
      path: "/AllUsers",
      name: "AllUsers",
      component: AllUsers
    }
  ],

  const router = new VueRouter({
    routes
  });

  export default router;

```

أوضحنا قبل قليل، أنَّ وظيفة الملف routes.js هي تعريف مسارات العناوين الداخلية التي سنستخدمها ضمن التطبيق. لننظر الآن إلى القسم الخاص بتعليمات الاستيراد:

```

import Vue from "vue";
import VueRouter from "vue-router";
import Home from "./views/Home";
import AddUser from "./views/AddUser";
import EditUser from "./views/EditUser";
import AllUsers from "./views/AllUsers";

```

أعتقد أنَّ أول تعليمتين واضحتين أما بالنسبة للتعليمات الأربع الأخرى، فهي لاستيراد الواجهات التي سنستخدمها في التطبيق. هذه الواجهات هي بطبيعة الحال مكوّنات (لأنّها موجودة ضمن ملفات تحمل الامتداد vue).

بعد ذلك نُخبر Vue.js أن يستخدم التوجيه من خلال التعليمات التالية:

```
Vue.use(VueRouter);
```

وبعد ذلك نصل إلى القسم الخاص بتعريف المسارات:

```

const routes = [
  {
    path: "/",

```

```

    name: "Home",
    component: Home
  },
  {
    path: "/AddUser",
    name: "AddUser",
    component: AddUser
  },
  {
    path: "/EditUser",
    name: "EditUser",
    component: EditUser
  },
  {
    path: "/AllUsers",
    name: "AllUsers",
    component: AllUsers
  },
];

```

لاحظ معي أننا نضع المسارات التي سنستخدمها في التطبيق ضمن مصفوفة اسمها `routes`. كل عنصر من هذه المصفوفة عبارة عن كائن له الحقول: `path` و `name` و `component`. انظر مثلاً إلى الكائن الأول من هذه المصفوفة:

```

{
  path: "/",
  name: "Home",
  component: Home
}

```

يمثل هذا الكائن المسار الخاص بالصفحة الرئيسية للتطبيق. فهو يُعرّف المسار الخاص بها ليكون `/` ويمثل الصفحة الرئيسية في تطبيقات الويب عادة. أي يمكن الوصول إلى هذه الواجهة من خلال الرابط التالي:

```
http://www.yourdomain.com/#/
```

أما اسم هذا المسار فهو `Home` (من الممكن أن نصل إلى أي مسار من خلال اسمه كما سنرى لاحقاً في هذا الفصل). بالنسبة للواجهة التي سترتبط بهذا المسار فهي `Home` (انظر إلى تعليمة الاستيراد الخاصة بالواجهة `Home.vue`).

بالنسبة لباقي المسارات فتعمل بنفس الأسلوب تماماً، فإذا أخذنا مثلاً المسار الثاني فنجد أن المسار الذي يُشير إليه هو `/AddUser` أي أننا نستطيع الوصول إليه عن طريق رابط مماثل لما يلي:

```
http://www.yourdomain.com/#/AddUser
```

نأتي إلى القسم التالي في الشيفرة البرمجية السابقة، حيث سنعرّف متغيّر جديد اسمه `router` ونسند إليه كائن جديد يمثل الموجه الذي سيدير عمليات التوجيه ضمن التطبيق:

```
const router = new VueRouter({
  routes
});
```

لاحظ كيف مرّرنا المصفوفة التي عرفنا ضمنها المسارات توجّهًا إلى `VueRouter` (وهو معرّف ضمن إحدى تعليمات الاستيراد). وأخيرًا نضع التعليمة المسؤولة عن تصدير الموجه `router` خارج هذا الملف.

لننتقل الآن إلى الملف `main.js`:

```
import Vue from 'vue'
import App from './App.vue'
import router from './routes.js'
import "bootstrap/dist/css/bootstrap.min.css";
import VueResource from 'vue-resource';
import VueSimpleAlert from "vue-simple-alert";

Vue.config.productionTip = false

Vue.use(VueResource);
Vue.use(VueSimpleAlert);

new Vue({
  router,
  render: h => h(App),
}).$mount('#app')
```

نستورد ضمن هذا الملف المكتبات التي سنحتاج إليها ضمن التطبيق. أعتقد أنّ أول تعليمتي استيراد واضحتين أيضًا. بالنسبة للتعليمة الثالثة، فهنا نستورد الموجه `router` التي بنيناها مسبقًا ضمن الملف `routes.js`، أمّا بالنسبة للتعليمات الثلاث الباقية فهي على الترتيب:

- استيراد مكتبة `Bootstrap` للتنسيق.
- استيراد المكتبة `vue-resource` للاتصال بالخواديم البعيدة، والتي تعاملنا معها في الفصل السابق.
- استيراد مكتبة جديدة اسمها `vue-simple-alert` لم نتعامل معها مسبقًا. والهدف منها عرض رسائل ذات تنسيق جميل للمستخدم، سنتعلّم بعد قليل كيفية التعامل معها.

سنحتاج بالتأكيد إلى تثبيت المكتبة `vue-simple-alert` بتنفيذ الأمر التالي ضمن موجه الأوامر:

```
npm install vue-simple-alert
```



لاحظ الآن التعليمتين التاليتين:

```
Vue.use(VueResource);
Vue.use(VueSimpleAlert);
```

هاتين التعليمتين لإخبار Vue.js أن يستخدم الكائنين: `VueResource` و `VueSimpleAlert`. نأتي الآن إلى آخر تعليمة ضمن هذا الملف:

```
new Vue({
  router,
  render: h => h(App),
}).$mount('#app')
```

التعليمة السابقة مألوفة، وقد استخدمنا مثلها مرارًا، ولكن لاحظ معي هنا كيف أضفت كائن لموجه `router` إليها. هذه الإضافة ضرورية، ولن يعمل التوجيه في التطبيق ما لم نضف كائن الموجه بهذه الصورة.

سنستعرض فيما يلي الملف `App.vue` بالإضافة إلى ملفات واجهات التطبيق.

## 11.2.1 الملف `App.vue`

ستكون محتويات هذا الملف على الشكل التالي:

```
<template>
  <div id="app">
    <nav id="nav">
      <p class="logo-place">Vue.js Remote Server (Enhanced)</p>
      <ul class="nav-links">
        <li class="links">
          <router-link to="/">Home</router-link>
        </li>
        <li class="links">
          <router-link to="/AddUser">Add User</router-link>
        </li>
        <li class="links">
          <router-link to="/AllUsers">All User</router-link>
        </li>
      </ul>
    </nav>
    <router-view></router-view>
  </div>
</template>

<script>
export default {
  name: "App",
```

```

    components: {},
  };
</script>

<style>

#nav {
  display: flex;
  margin-bottom: 24px;
}

#nav a {
  font-weight: bold;
  color: #2c3e50;
  text-decoration: none;
  margin-right: 12px;
}

#nav a.router-link-active {
  color: #ab26ab;
}

.nav-links {
  padding-right: 20px;
  list-style: none;
  display: flex;
  margin: 21px 0 0 0;
}

.links-hover {
  text-decoration: underline;
}

.logo-place {
  font-size: 20px;
  color: purple;
  font-weight: bold;
  margin: 16px 0 0 16px;
}
</style>

```

هناك أمران جديان سنتحدث عنهما بالنسبة لهذا الملف. لاحظ أولاً وجود وسم جديد اسمه `router-view`. في الحقيقة أنّ هذا الوسم ما هو إلاّ مكون موجود ضمن المكتبة `vue-router`، ووظيفته تحديد مكان إدراج واجهات التطبيق المختلفة ضمن الصفحة الرئيسية. لأنّ تطبيقات SPA كما أشرنا من قبل، لا تتطلب إعادة تحميل الصفحة بالكامل عندما يريد المستخدم الانتقال إلى واجهة مختلفة.

الأمر الآخر هو وجود وسم جديد آخر وهو `router-link` وهو مكون أيضًا ضمن نفس المكتبة. وظيفة هذا المكون توليد عنصر ارتباط تشعبي `<a>` بحيث ينتقل إلى إحدى الواجهات المعرّفة ضمنه. فمثلاً لاحظ معي المكون `router-link` التالي التي أخذته من الشيفرة السابقة:

```
<router-link to="/AddUser">Add User</router-link>
```

نجد من المقطع السابق وجود السمة `to` وهي مسؤولة عن تحديد المسار (Path) الذي سيؤجّه إليه المستخدم عند نقر هذا الرابط. في المثال السابق سيؤجّه المستخدم إلى المسار الكامل التالي:

```
http://www.yourdomain.com/#/AddUser
```

وذلك بسبب وجود `AddUser` كقيمة للسمة `to`.

## 11.2.2 الملف `Home.vue`

وهو ملف الصفحة الرئيسية في التطبيق وستكون محتوياته على الشكل التالي:

```
<template>
  <div class="container">
    <div class="row">
      <div class="col-6 offset-2">
        <h1>Welcome in our website!</h1>
      </div>
    </div>

    <div class="row" style="margin-top:30px;">
      <div class="card" style="width: 18rem;">
        <div class="card-body">
          <h5 class="card-title">Add User</h5>
          <p class="card-text">Add a new user to the Firebase
experimental website.</p>
          <router-link class="btn btn-primary" to="/AddUser">Add a
user</router-link>
        </div>
      </div>

      <div class="card" style="width: 18rem; margin-left:18px;">
        <div class="card-body">
          <h5 class="card-title">All User</h5>
          <p class="card-text">Display all users info from the
Firebase experimental website.</p>
          <router-link class="btn btn-primary" to="/AllUsers">Get all
users</router-link>
        </div>
      </div>
    </div>
  </div>
```

```

</template>

<script>
export default {
  name: "Home",
};
</script>

```

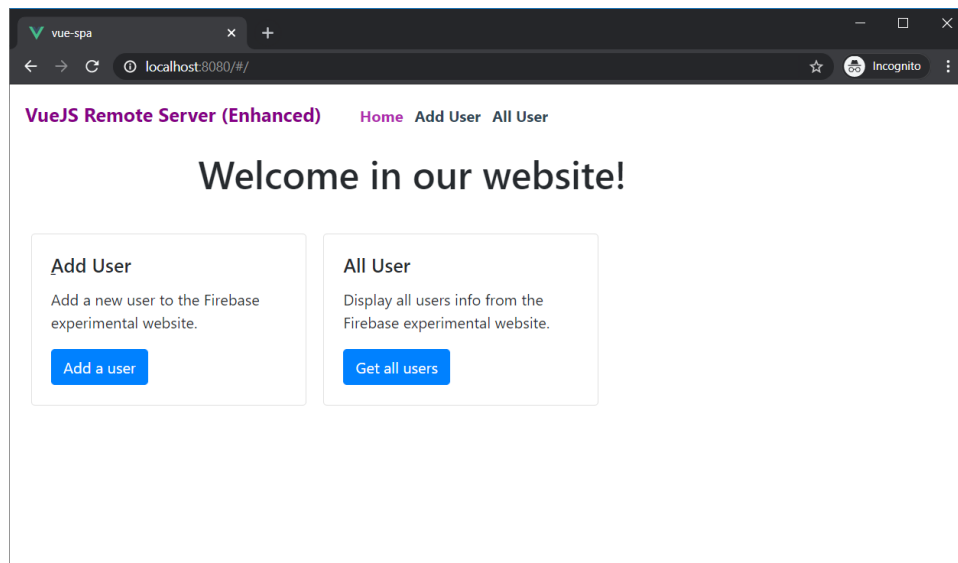
لا يوجد في الشيفرة السابقة أي جديد، سوى أنك تمتلك الحق في إضافة أصناف تنسيقية إلى المكوّن router-link كما في:

```

<router-link class="btn btn-primary" to="/AddUser">Add a user</router-link>

```

ستبدو هذه الواجهة على النحو التالي:



### 11.2.3 الملف AddUser.vue

يحتوي هذا الملف على الواجهة المسؤولة عن إضافة مستخدم جديد. انظر معي إلى الشيفرة البرمجية الخاصة به:

```

<template>
  <div class="container">
    <div class="row">
      <div class="col-6 offset-2">
        <h1>Add User</h1>
      </div>
    </div>
    <div class="row">
      <div class="col-6 offset-2">
        <div class="container">

```

```

        <div class="row">
          <div class="col-12">
            <div class="form-group">
              <label>Username</label>
              <input type="text" class="form-control" v-
model="user.username" />
            </div>
            <div class="form-group">
              <label>First name</label>
              <input type="text" class="form-control" v-
model="user.firstname" />
            </div>
            <div class="form-group">
              <label>Last name</label>
              <input type="text" class="form-control" v-
model="user.lastname" />
            </div>
          </div>
          <div>
            <button class="btn btn-primary float-left" style="margin-
left:12px;" @click="postUser()">Add</button>
          </div>
        </div>
      </div>
    </template>

    <script>
    export default {
      name: "AddUser",
      methods: {
        postUser() {
          this.$http
            .post("https://vue-remote-servers.firebaseio.com/users.json",
this.user)
            .then(
              () => {
                this.user.username = "";
                this.user.firstname = "";
                this.user.lastname = "";

                this.$alert("A new user is added.", "Operation Succeeded",
"success");
              },
              (error) => {
                console.log(error);
              }
            );
        },
      },
      data() {

```

```

    return {
      user: {
        username: "",
        firstname: "",
        lastname: "",
      },
    };
  },
};
</script>

<style scoped>
.row {
  margin-top: 8px;
}
</style>

```

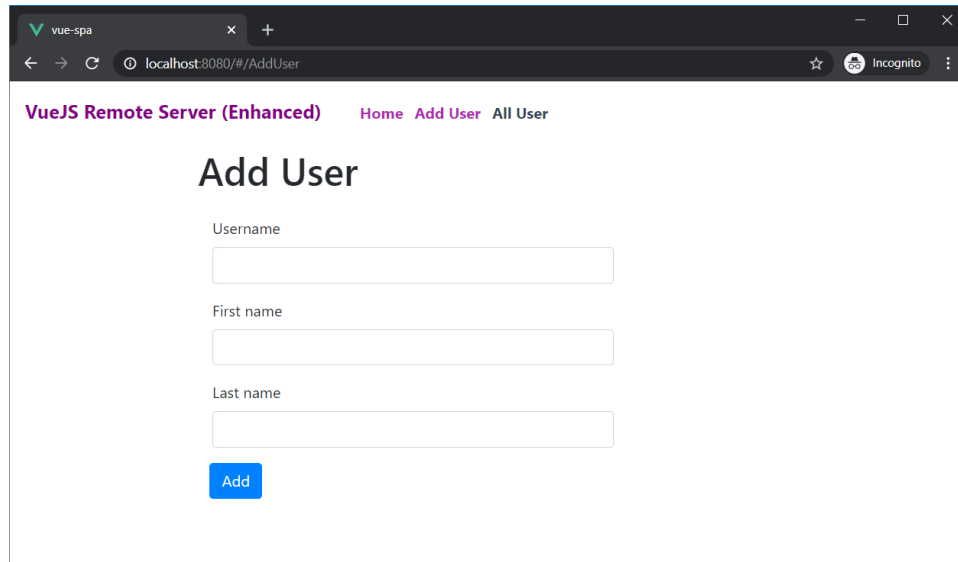
الشفيرة البرمجية الموجودة ضمن هذا الملف تماثل تقريبًا تلك التي كانت مسؤولة عن إضافة مستخدم جديد في الفصل الماضي. مع ملاحظة أنَّ الشيفرة أصبحت أكثر تنظيماً، بسبب أننا نعزل هنا عملية إضافة مستخدم جديد عن غيرها من العمليات.

على أية حال توجد ميزة جديدة استُخدمت ضمن هذه الواجهة، وهي رسالة ستُعرض للمستخدم في حال نجاح عملية الإضافة إلى قاعدة بيانات Firebase:

```
this.$alert("A new user is added.", "Operation Succeeded", "success");
```

كما كان من الممكن إضافة رسالة شبيهة بالرسالة السابقة في حال فشلت عملية الإضافة للمستخدم (لم أضف مثل هذه الرسالة إلى الشيفرة السابقة). التابع `$alert` موجود ضمن المكتبة `vue-simple-alert` وظيفته كما هو واضح، هو في عرض رسالة منسقة للمستخدم. بالنسبة لمثالنا هذا، سنعرض رسالة تُخبر المستخدم بأنَّ عملية إضافة بيانات المستخدم إلى قاعدة بيانات Firebase قد تمت بنجاح. يمثل الوسيط الأول للتابع `$alert` الرسالة التي نريد عرضها، أما الوسيط الثاني فهو عنوان صندوق الرسالة، أما الوسيط الثالث فهو نوع الرسالة وهي في حالتنا هذه `success`. توجد أنواع أخرى للرسائل التي يمكن للتابع `$alert` عرضها مثل `success` و `error`. للاطلاع على المزيد من الوثائق المتعلقة بهذه المكتبة يمكنك زيارة [الصفحة الرسمية لها](#).

ستبدو هذه الواجهة على النحو التالي:



## 11.2.4 الملف AllUsers.vue

يحتوي هذا الملف على الواجهة الخاصة بعرض بيانات المستخدمين الذين سبق إضافتهم إلى قاعدة بيانات Firebase. انظر إلى الشيفرة البرمجية الخاصة به:

```
<template>
  <div class="container">
    <div class="row">
      <h2>All Users</h2>
      <button style="margin-left:16px;" class="btn btn-primary"
@click="getData()">Retrieve</button>
    </div>
    <hr />
    <div class="row" v-for="usr in users" v-bind:key="usr.username">
      <div class="col-2">{{usr.username}}</div>
      <div class="col-4">{{usr.firstname}} {{usr.lastname}}</div>
      <div class="col-1">
        <router-link :to="{name: 'EditUser', params:
{user:usr}}">Edit</router-link>
      </div>
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      user: {
        username: "",
        firstname: "",
        lastname: "",
      },
    },
  },
}
```

```

      users: [],
    };
  },
  methods: {
    getData: function () {
      this.$http
        .get("https://vue-remote-servers.firebaseio.com/users.json")
        .then((response) => {
          return response.json();
        })
        .then((data) => {
          const tmpArray = [];

          for (let key in data) {
            let withId = data[key];
            withId.id = key;
            tmpArray.push(data[key]);
          }

          this.users = tmpArray;
        });
    },
  },
};
</script>

<style scoped>
.row{
  margin-top:8px;
}
</style>

```

مرة أخرى، لا تختلف الشيفرة البرمجية الموجودة هنا عن تلك التي كانت موجودة في الفصل السابق. فعندما ينتقل المستخدم لهذه الواجهة وينقر على الزر Retrieve، سثدث الواجهة بحيث تعرض جميع المستخدمين المخزنين حالياً ضمن قاعدة بيانات Firebase. بعد عرض هؤلاء المستخدمين، ستلاحظ وجود رابط اسمه Edit يظهر بجوار كل مستخدم. عند نقر هذا الزر سينقلك التطبيق إلى واجهة تحرير بيانات المستخدم EditUser لإجراء التعديلات المطلوبة عليه.

ولكن السؤال هنا، كيف ستمكن الواجهة EditUser من معرفة المستخدم المطلوب تحرير بياناته؟ الجواب هو في المكون router-link الموجود ضمن هذه الواجهة AllUsers. انظر معي إلى الاستخدام الجديد للمكون router-link:

```

<router-link :to="{name: 'EditUser', params:
{user:usr}}">Edit</router-link>

```



استخدمنا في المقطع السابق السمة `to` بشكل مختلف عن استخدامنا لها في الواجهات السابقة. لاحظ أولاً كيف وضعت نقطتين رأسيين قبل كلمة `to` مباشرةً (وهما ضروريتان في هذه الحالة)، ثم أسندت إلى السمة `to`: كائن وليس نص عادي كما فعلنا من قبل:

```
{name: 'EditUser', params: {user:usr}}
```

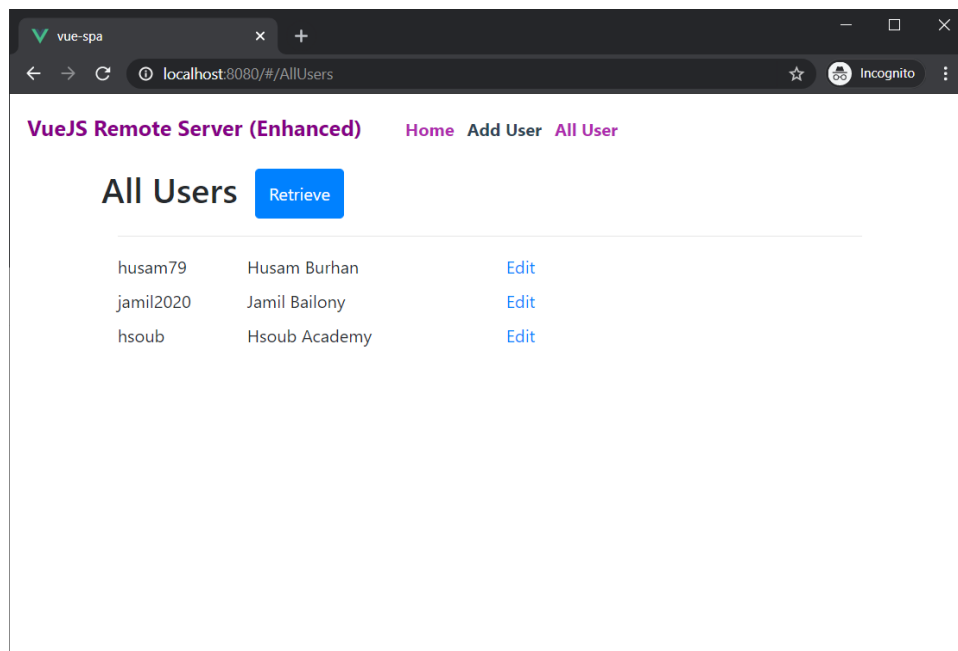
هذا الكائن لديه مفتاحين. الأول هو `name` ويمتلك القيمة `'EditUser'`، ويمثل اسم الواجهة التي نريد الانتقال إليها عندما ينقر المستخدم هذا الرابط. لاحظ أنني قلت "اسم" وليس "مسار"، أي أن الاسم الذي نكتبه هنا يجب أن يكون موافقاً للقيمة `name` التي كانت موجودة ضمن ملف المسارات `routes.js` عندما عرفنا تلك المسارات إذا كنت تذكر.

بالنسبة للمفتاح الثاني فهو `params` وهو مسؤول عن تمرير أية وسائط نرغبها إلى الواجهة التي نريد الانتقال إليها. في حالتنا هذه نرغب بتمرير بيانات المستخدم المراد تحريرها بشكل كامل. طريقة التمرير مفيدة للغاية، لاحظ كيف مَرَرنا كائن جديد للمفتاح `params` يحتوي هذا الكائن في حالتنا هذه، على مفتاح وحيد اسمه `user` ونسند له القيمة `usr` التي تُعتبر كائنًا مستقلًا بحد ذاته يحتوي على بيانات المستخدم المراد تحرير بياناته. أي أن `{user:usr}` عبارة عن ثنائية تحوي اسم الوسيط `user` مع القيمة المرتبطة به `usr`.

يمكن بالطبع تمرير أكثر من وسيط إلى الواجهة بالأسلوب السابق، وذلك بوضعها بجوار بعضها على شكل مفاتيح متعاقبة كما يلي:

```
params: {key1:val1, key2:val2, ...}
```

ستبدو الواجهة `AllUsers` على النحو التالي:



## 11.2.5 الملف EditUser.vue

في هذه الواجهة يمكننا تعديل بيانات المستخدم، بالإضافة إلى إمكانية حذف بيانات هذا المستخدم بشكل كلي من قاعدة البيانات (هذه ميزة جديدة لم تكن مضافة مسبقًا في الفصل السابق).

إليك الشيفرة البرمجية الخاصة بهذا الملف:

```
<template>
  <div class="container">
    <div class="row">
      <div class="col-6 offset-2">
        <h1>Edit User</h1>
      </div>
    </div>
    <div class="row">
      <div class="col-6 offset-2">
        <div class="container">
          <div class="row">
            <div class="col-12">
              <div class="form-group">
                <label>Username</label>
                <input type="text" class="form-control" v-
model="user.username" />
              </div>
              <div class="form-group">
                <label>First name</label>
                <input type="text" class="form-control" v-
model="user.firstname" />
              </div>
              <div class="form-group">
                <label>Last name</label>
                <input type="text" class="form-control" v-
model="user.lastname" />
              </div>
            </div>
          </div>
          <div>
            <button class="btn btn-primary float-left" style="margin-
left:12px;" @click="putUser()">Save</button>
            <button
              class="btn btn-danger float-right"
              style="margin-right:12px;"
              @click="deleteUser()"
            >Delete</button>
          </div>
        </div>
      </div>
    </div>
  </template>
```

```
<script>
export default {
  name: "EditUser",
  methods: {
    putUser() {
      this.$http
        .put(
          "https://vue-remote-servers.firebaseio.com/users/" +
            this.$route.params.user.id +
            ".json",
          this.user
        )
        .then(
          () => {
            this.$alert(
              "The user is updated.",
              "Operation Succeeded",
              "success"
            );
          },
          (error) => {
            console.log(error);
          }
        );
    },
    deleteUser() {
      this.$confirm("Are you sure you want to delete the
user?").then(() => {
        this.$http
          .delete(
            "https://vue-remote-servers.firebaseio.com/users/" +
              this.$route.params.user.id +
              ".json"
          )
          .then(
            () => {
              this.$router.push('AllUsers');
            },
            (error) => {
              console.log(error);
            }
          );
      });
    },
  },
  computed: {
    user() {
      return this.$route.params.user;
    },
  },
};
```

```

</script>

<style scoped>
.row {
  margin-top: 8px;
}
</style>

```

الشيء الذي أود التركيز عليه من الشيفرة السابقة هو كيفية استلام الوسائط الممّزة لهذه الواجهة من الواجهة الخاصة بعرض بيانات جميع المستخدمين AllUsers ومن ثمّ تعبئة هذه البيانات ضمن حقول البيانات المناسبة لها، ليتمكن المستخدم من تعديلها إن أحب. أنشأت لهذا الغرض خاصية محسوبة اسمها user إليك الشيفرة البرمجية الخاصة بها:

```

user() {
  return this.$route.params.user;
}

```

تحتوي هذه الخاصية المحسوبة على تعليمة برمجية واحدة تزجج الوسيط الذي مّررناه من الواجهة AllUsers قبل قليل:

```

this.$route.params.user

```

الوسيط هو user كما أسميناه ضمن الواجهة AllUsers، يحتوي هذا الوسيط على بيانات المستخدم username و firstname و lastname كما وردت من الواجهة AllUsers. هذا الوسيط موجود ضمن المفتاح params كما هو واضح، والموجود بدوره ضمن الكائن \$route (الموجود ضمن المكتبة vue-router). وبما أنّ الخاصية user محسوبة فإنّ البيانات المستخلصة منها سنعقّم مباشرة على الحقول الثلاثة الموجودة ضمن الواجهة EditUser كما يلي:

```

<div class="col-12">
  <div class="form-group">
    <label>Username</label>
    <input type="text" class="form-control" v-
model="user.username" />
  </div>
  <div class="form-group">
    <label>First name</label>
    <input type="text" class="form-control" v-
model="user.firstname" />
  </div>
  <div class="form-group">
    <label>Last name</label>
    <input type="text" class="form-control" v-
model="user.lastname" />
  </div>
</div>

```

```
</div>
</div>
```

أود أيضًا التحدث عن التابع `$confirm` الموجود ضمن المكتبة `vue-simple-alert` ووظيفته تذكيرك بين أمرين. استخدمت التابع `$confirm` ضمن التابع `deleteUser` وذلك لكي نطلب من المستخدم تأكيد أنه يريد حذف المستخدم الحالي من قاعدة البيانات. انظر الشيفرة البرمجية للتابع `deleteUser`:

```
this.$confirm("Are you sure you want to delete the user?").then(() =>
{
  this.$http
    .delete(
      "https://vue-remote-servers.firebaseio.com/users/" +
      this.$route.params.user.id +
      ".json"
    )
    .then(
      () => {
        this.$router.push('AllUsers');
      },
      (error) => {
        console.log(error);
      }
    );
});
```

في حال تمت الموافقة على حذف المستخدم الحالي، سيقفز تابع السهم المُمرر لتابع `then` الأول. يحتوي تابع السهم هذا على الشيفرة البرمجية التي ستحذف بيانات المستخدم من قاعدة بيانات `Firebase` وهي مألوفة وتشبه أخواتها من المقاطع البرمجية الأخرى المسؤولة عن التعامل مع قاعدة بيانات `Firebase`. لاحظ أننا ننفذ تابع `then` آخر يحتوي على تعليمة برمجية أخرى وظيفتها تحويل المستخدم إلى الواجهة `AllUsers` بعد الانتهاء من حذف المستخدم الحالي. انظر لهذه التعليمة:

```
this.$router.push('AllUsers');
```

هذا هو الأسلوب المتبع للانتقال بين الواجهات بشكل برمجي، وذلك عن طريق استخدام التابع `push` من الكائن `$router`، حيث نمرّر "اسم" الواجهة التي نرغب بالانتقال إليها كوسيط للتابع `push`.

## 11.3 ختام الفصل

تناولنا في هذا الفصل مقدّمة إلى التوجيه في Vue.js. في الحقيقة يعد هذا الموضوع كبيرًا بعض الشيء وله جوانب متعدّدة غطينا في هذا الفصل الأساسي منها. في حال احتجت إلى التوسّع في هذا الموضوع مستقبلاً، فأعتقد أنّ خير رفيق لك سيكون التوثيق الرسمي للتوجيه تجده على [هذا الموقع](#) ولكن باللغة الإنجليزية. أقترح عليك بعد قراءة هذا الفصل أن تحاول تجريب بناء تطبيقات بسيطة تعتمد على التوجيه، لكي تعتاد على هذه التقنية.

# 12. نشر تطبيق Vue.js إلى الإنترنت

سنتعلم في هذا الفصل:

- تجهيز التطبيق قبل النشر
- إنشاء موقع جديد على منصة Netlify بالاستناد إلى مستودع GitHub

هذا الفصل هو الأخير، حيث سنتّوج هذا الكتاب بشرح كيفية نشر التطبيق الوارد في الفصل السابق، وهو عبارة عن تطبيق SPA يدعم التخاطب مع قاعدة بيانات موجودة على Firebase كما تذكر. بعد بناء التطبيق ونشره، سيتمكّن أي مستخدم حول العالم من الوصول إلى تطبيقنا هذا والتفاعل معه. سأعتمد استخدام المنصة المجانية Netlify لهذا الغرض، ويمكنك بالطبع استخدام أي منصة أو خادم مخصص ترغب به.

## 12.1 تجهيز التطبيق قبل النشر

كما ذكرت قبل قليل، سنعتمد التطبيق المبني في الفصل السابق لتشغيله على خدمة Netlify. بغية ذلك، عملت على رفع هذا التطبيق بشكل كامل على مستودع Github التالي:

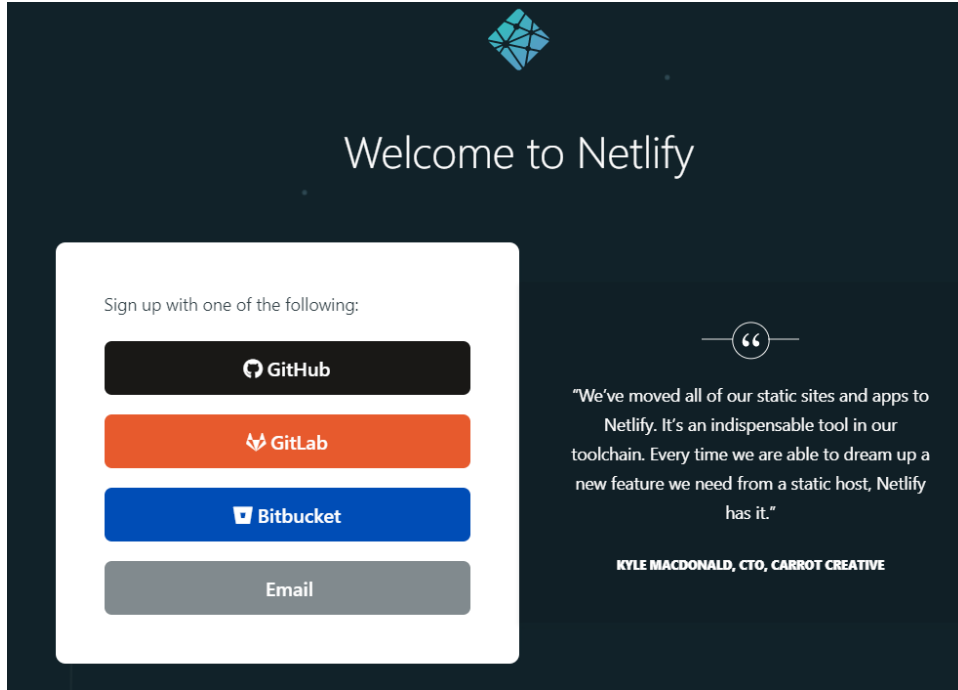
[github.com/HsoubAcademy/vuejs-spa](https://github.com/HsoubAcademy/vuejs-spa)

سبب ذلك هو أنّ Netlify يسحب التطبيقات الموجودة على خدمات Git مثل GitHub. ستحتاج بالطبع إلى أن يكون لديك حساب على GitHub. زُر المستودع السابق، وسجّل دخولك على GitHub إذا لم تكن قد فعلت ذلك مسبقاً، ومن الصفحة الرئيسية لمستودعنا، انقر الزر Fork في الزاوية اليمنى العليا لإنشاء تفرعة جديدة من الشيفرة البرمجية الحالية.

سيطلب الأمر لحظات حتى يتم إنشاء التفرعة من الشيفرة البرمجية ووضعها ضمن حسابك الخاص.

## 12.2 نشر التطبيق على منصة Netlify

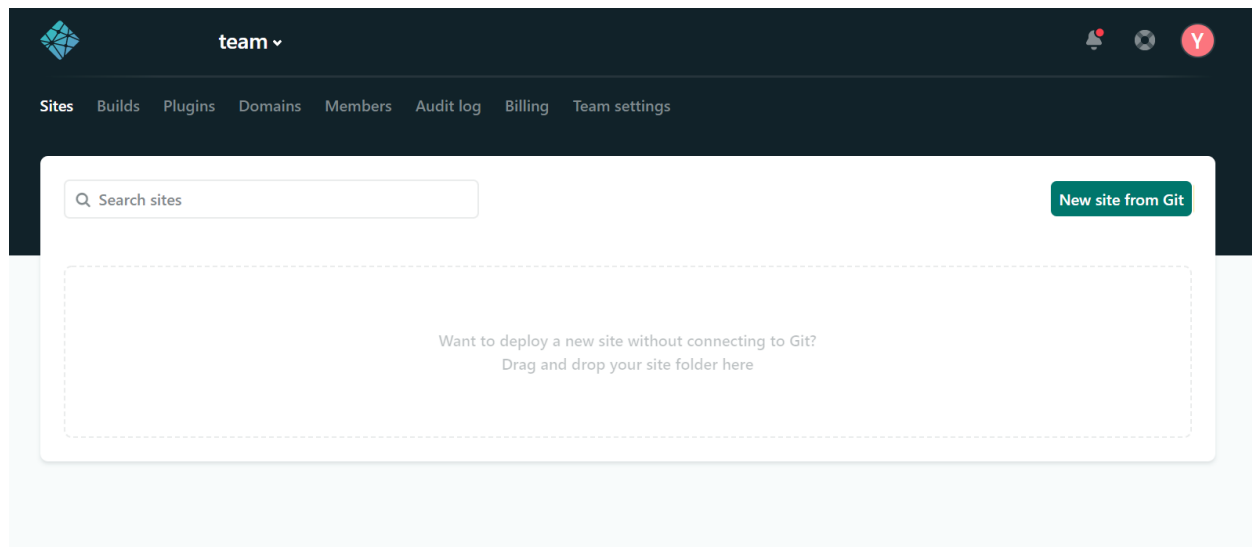
أنشئ حسابًا جديدًا على Netlify عن طريق زيارة الصفحة الرئيسية له [الموقع الرسمي](#). انقر Sign up من الزاوية اليمنى العليا لتظهر لك الصفحة الخاصة بإنشاء اشتراك جديد. تبدو هذه الصفحة حاليًا على النحو التالي:



لاحظ معي وجود خيارات متعددة للتسجيل. سنستخدم في حالتنا هذه التسجيل باستخدام البريد الإلكتروني. انقر الخيار الأخير "Email".

بعد كتابة البريد الإلكتروني الذي تودّ التسجيل عن طريقه، سيرسل الموقع رسالة تحقق إلى هذا البريد تحتوي على رابط لتفعيل الحساب الجديد، انقر ذلك الرابط المرسل إلى بريدك، وبعد تسجيل الدخول إلى Netlify باستخدام حسابك الجديد ستظهر صفحة مشابهة لما يلي:





هذه هي الصفحة الرئيسية والتي تحتوي على جميع عناصر التحكم الخاصة بحسابك. نحن الآن موجودون ضمن الصفحة الخاصة بمواقع الويب التي أنشأتها أنت مسبقًا، وهي فارغة الآن بطبيعة الحال. انقر الزر "New site from Git" من الطرف الأيمن في الأعلى، للاتصال بمستودع من النوع Git. ستحصل على شكل شبيه بما يلي:

## Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider

2. Pick a repository

3. Build options, and deploy!


## Continuous Deployment

Choose the Git provider where your site's source code is hosted. When you push to Git, we run your build tool of choice on our servers and deploy the result.

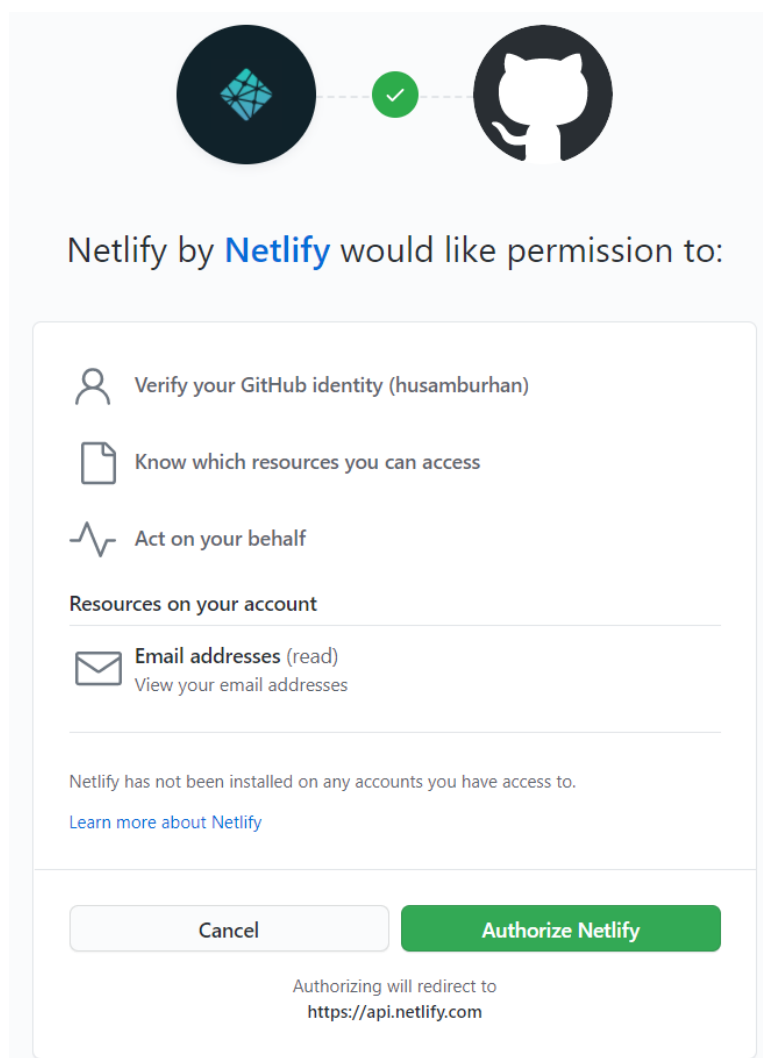
You can [unlock options for self-hosted GitHub/GitLab](#) by upgrading to the Business plan.

 GitHub

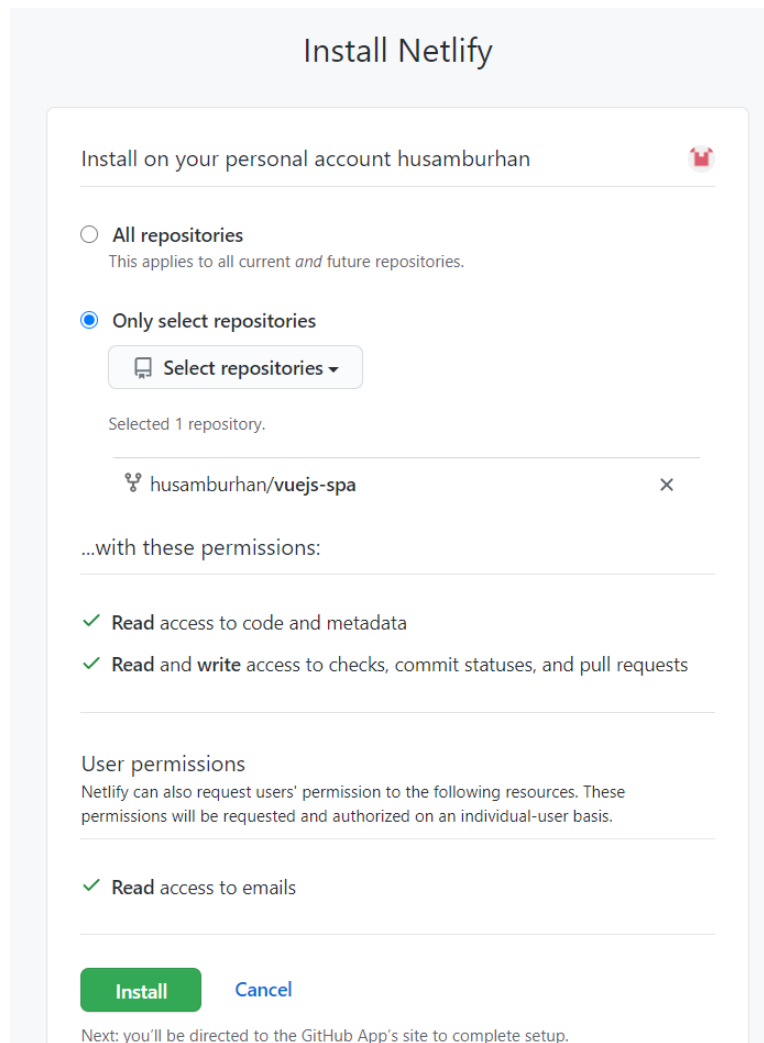
 GitLab

 Bitbucket

انقر الزر GitHub الموجود في الأسفل، سيؤدي ذلك إلى فتح نافذة جديدة قد تطلب منك تسجيل الدخول بحسابك على GitHub في حال لم تكن مسجلًا دخولك على GitHub بعد (ذلك الحساب الذي أنشأت التفرعة ضمنه):



انقر الزر **Authorize Netlify** للسماح لـ Netlify بالوصول إلى حساب GitHub الخاص بك. بعد إكمال عملية الاستيثاق (Authorization)، ستظهر لك صفحة تسمح لك بالاختيار بين السماح الكامل لـ Netlify للوصول إلى جميع المستودعات ضمن حسابك أو السماح بالوصول إلى مستودع محدد. بالنسبة إلي، سأسمح له بالوصول إلى المستودع vuejs-spa فقط، وهو المستودع الذي أنشأناه كتفريعة قبل قليل من الفقرة السابقة.



انقر زر Install لتبدأ عملية التثبيت التي تتضمن الانتقال إلى موقع GitHub بشكل تلقائي وذلك لإكمال عملية الربط (قد يطلب منك كلمة المرور لحسابك في GitHub مرة أخرى).

بعد الانتهاء من العملية السابقة ستحتاج إلى العودة إلى الصفحة الرئيسية لموقع Netlify وتكرار عملية إنشاء موقع من مستودع GitHub (الشكل رقم 3). هذه المرة سيظهر لنا المستودع الذي نرغب باستيراده. انظر إلى الشكل التالي:

## Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider 2. Pick a repository 3. Build options, and deploy!

### Continuous Deployment: GitHub App

Choose the repository you want to link to your site on Netlify. When you push to Git, we run your build tool of choice on our servers and deploy the result.


husamburhan ▾


husamburhan/vuejs-spa

Can't see your repo here? [Configure the Netlify app on GitHub.](#)

لاحظ معي كيف ظهر المستودع vuejs-spa مع اسم حساب GitHub التابع لي husamburhan. سأختار هذا المستودع بالنقر عليه، سيؤدي ذلك إلى الخطوة النهائية والتي تتمثل بتحديد الفرع الرئيسي (Branch) التي سنستخدمه في عملية نشر التطبيق، بالإضافة إلى تحديد التعليمات الخاصة ببناء التطبيق، والمجلد الخاص بالنشر. احرص على أن تحصل على شكل شبيه بما يلي:

### Deploy settings for husamburhan/vuejs-spa

Get more control over how Netlify builds and deploys your site with these settings.

Owner

Husam Burhan
▾

Branch to deploy

master
▾

### Basic build settings

If you're using a static site generator or build tool, we'll need these settings to build your site.

[Learn more in the docs](#)

Build command

npm run build

Publish directory

/dist

Show advanced

Deploy site

أخيرًا، انقر الزر Deploy site لبناء التطبيق ونشر الموقع على الإنترنت. ستأخذ هذه العملية القليل من الوقت، حيث ستنقل إلى صفحة تخبرك بأن عملية البناء والنشر قيد التجهيز. بعد أن تنتهي عملية البناء والنشر ستحصل على رابط حي للصفحة تستطيع مشاركته مع من ترغب.

بالنسبة لي حصلت على الرابط التالي: [suspicious-mahavira-10e9ac.netlify.app](https://suspicious-mahavira-10e9ac.netlify.app)

لاحظ أن اسم الموقع يُولّد بشكل عشوائي. يمكنك بالطبع تغيير هذا الاسم بحيث يصبح منطقيًا أكثر. كما يمكنك إضافة اسم نطاق مخصّص مثل [www.example.com](https://www.example.com) وربطه مع هذا الموقع إن أحببت. جميع الخيارات السابقة وأكثر يمكن التحكم بها من خلال خيارات الموقع في Netlify.

## 12.3 ختام الفصل

ختمنا في هذا الفصل كتابنا هذا، أساسيات إطار العمل Vue.js، بشرح كيفية نشر التطبيق إلى خدمة مجانية ليصبح تطبيقنا متاحًا على الإنترنت. توجد العديد من الخدمات التي تسمح بالنشر المجاني، ولكنني اخترت أبسطها وهي Netlify حسب رأيي.

حاولت في هذا الكتاب تغطية الأمور الأساسية فقط في Vue.js، لذلك فأمامك الكثير بكل تأكيد لتتعلمه حول هذه التقنية الواعدة. يمكنك زيارة [الموقع الرسمي](#) لإطار العمل Vue.js والاستزادة حوله.

# أحدث إصدارات أكاديمية حسوب

