

I love Vue

Creando una pequeña aplicación con Vue.js

Celia Amador @celtwine

Presentación

Agradecimientos

- WomanTechMaker por organizar el evento.
- APSL por el patrocinio
- A los asistentes por la gran acogida que ha tenido el taller.

Sobre mí

- Programadora desde 2007 / 2008
- Muy centrada en front-end desde 2014-2015
- He trabajado con los principales frameworks (Angular 1, React, y ahora...Vue.js)

Disclaimer...

- He aprendido Vue.js de forma más o menos autónoma
- Las bases de los diferentes frameworks de front-end son muy similares entre sí, por lo que pasar de uno a otro no supone una gran dificultad.
- No vamos a ver en el taller todo lo relevante de Vue.js, aunque se han incluido gran parte de las nociones básicas del framework

...pero tampoco es esto



1. Creación del proyecto

- 1.1) Creamos una carpeta “vue-lab”. Dentro de ella, instalamos dos paquetes de npm, que nos permitirán crear el proyecto desde cero.

```
# Con npm (4.1 o superior):  
mkdir ~/vue-lab && cd ~/vue-lab  
npm install vue@2.6.8  
npm install @vue/cli@3.5.0
```

```
# Con yarn:  
mkdir ~/vue-lab && cd ~/vue-lab  
yarn add vue@2.6.8  
yarn add @vue/cli/3.5.0
```

- 1.2) Hacemos un clon del repositorio del proyecto y hacemos “checkout” de la rama base.

```
git clone https://github.com/celdrake/i-love-vue.git  
cd i-love-vue  
git checkout i-love-vue-commit-0
```

- 1.3) Creamos un nuevo proyecto de Vue.js

```
# Importante: mismo nombre de carpeta con el que esté bajado el repositorio  
cd ..  
./node_modules/.bin/vue create i-love-vue
```

1.1 Configuración del proyecto (I)

- Seleccionamos la configuración del proyecto. Ir con especial cuidado a la hora de seleccionar las opciones que se marcan con la barra espaciadora (tecla de Espacio), y no clicando “Enter”.
 - [Marcar con Enter] Merge



Pantalla para seleccionar la primera opción de configuración

1.1 Configuración del proyecto (II)

- Continuamos con el resto de opciones:
 - [Marcar con Enter] Manually select features
 - **[Marcar con ESPACIO]** CSS Preprocessors
 - [Marcar con Enter] Sass/CSS (with node sass)
 - [Marcar con Enter] Eslint with error prevention only
 - [Dejar activa] Lint on save
 - **[Marcar con ESPACIO]** Lint and fix on commit
 - [Marcar con Enter] In package.json
 - Save this template? NO

1.2 Comprobación del proyecto

- Arrancamos la aplicación que hemos creado, y vamos a <http://localhost:8080>

```
cd i-love-vue  
npm run serve ó yarn serve
```

- Arrancamos la aplicación gráfica que gestiona los proyectos de Vue, en <http://localhost:8000/dashboard>

```
cd ..  
./node_modules/.bin/vue ui
```

- Importamos el proyecto, desde <http://localhost:8000/project/select> (Import)

NOTA: Ahora podremos usar el menú “**Tasks / Tareas**” para arrancar y parar la aplicación desde la interfaz gráfica si no queremos hacerlo a través del comando “npm run serve”.

1.3 Extensiones Router y Vuex

- Es el momento de comitar los ficheros iniciales

Rama: i-love-vue-commit-1

```
git add . && git commit -m "1. Autogenerated by Vue create project"
```

- El fichero “main.js” se encarga de conectar Vue con nuestra aplicación, representada en App.vue
- Añadimos tres extensiones muy útiles, desde <http://localhost:8080/plugins>
 - “Add vue-router”
 - “Add vuex”
 - “Install devtools” (extensión del navegador)

1.4 Breve comentario sobre Router + Vuex

- Con **git status** podremos ver los cambios debidos a la adición del Router y Vuex.
- Vue-router nos permite definir distintas URLs con vistas (Componentes de Vue) distintos
- Vuex nos permite gestionar el estado de nuestra aplicación y sincronizar componentes entre sí.
- Comitamos los nuevos cambios

Rama: i-love-vue-commit-2

```
git add . && git commit -m "2. Added Router + Vuex. Ready to go!"
```

2. Programando MemoryMatrix!

- Nuestro proyecto está basado en el siguiente ejemplo:

<https://github.com/yoony12345678910/MemoryMatrix>

- Hemos simplificado el comportamiento con unas adaptaciones
- Más adelante plantearemos variaciones para hacerlo tan complicado como se quiera :)

2.1 Uso del repositorio git (opcional)

- A lo largo del taller, cada bloque de cambios se concreta en un commit, que dispone de una rama propia en el repositorio de Github
- Es decir, al finalizar el paso 2, se comitan todos los cambios en la rama

“i-love-vue-commit-2”

- Si alguien tiene dificultades en uno de los bloques, puede optar por bajarse la rama que contiene los cambios de dicho bloque para continuar con el taller.

Ejemplo para guardar los cambios hasta el momento y pasar a la rama base del paso 2:

```
git checkout -b paso-x-incompleto  
git add . && git commit -m "cambios durante el taller"  
git checkout i-love-vue-commit-x
```

Nota: hasta el paso 2, después de bajar la rama, deberéis hacer “**npm install**”

2.1 Definiendo los componentes

- Renombramos la vista “**Home**” a “**MemoryMatrix**”.
Cambiamos los imports hacia Home en “**router.js**”, y la ruta “**/home**” por “**/game**”
- Eliminamos el componente “**HelloWorld**”.
- Creamos dos nuevos componentes: “**GameInfo**” y “**GameBox**”. Los añadimos en **MemoryMatrix**:

GameInfo.vue y GameBox.vue: MemoryMatrix.vue

```
<template>
  <div class="game-info">
    Game info here
  </div>
</template>

<script>
  export default {
    name: 'GameInfo',
  }
</script>

<style lang="scss" scoped>
  .game-info {
    border: 1px solid gray;
  }
</style>
```

```
<template>
  <div class="memory-matrix">
    <game-info />
    <game-box />
  </div>
</template>

<script>
  import GameBox from '@components/GameBox.vue'
  import GameInfo from '@components/GameInfo.vue'

  export default {
    name: 'MemoryMatrix',
    components: {
      GameBox,
      GameInfo,
    }
  }
</script>
```

2.1 Creación del estado del tablero

- Comitamos los componentes

Rama: i-love-vue-commit-3

```
git add . && git commit -m "3. Create basic game layout"
```

- Definimos el estado inicialmente en “**GameBox**”
 - Creamos el método + data
 - Visualizamos en el template

```
initMatrix(gameSize) {  
  const matrix = [];  
  for (let row = 0; row < gameSize; row += 1) {  
    const rowColumns = [];  
    for (let col = 0; col < gameSize; col += 1) {  
      rowColumns.push({  
        display: true,  
        content: Date.now() % 1000,  
      });  
    }  
    matrix.push(rowColumns);  
  }  
  return matrix;  
},  
// En data, invocar como "this.initMatrix(4)"
```

```
<div v-for="(row, rowIndex) in matrix" :key="rowIndex">  
  <div v-for="(cell, cellIndex) in row" :key="cellIndex">  
    {{ cell.content }}  
  </div>  
</div>
```

2.2 Visualización del tablero (I)

Una capa de maquillaje:

Recordar añadir las clases!

```
.game-box {  
  display: flex;  
  align-items: center;  
  justify-content: center;  
  width: 100%;  
  flex-direction: column;  
  &__row {  
    display: flex;  
    flex-wrap: nowrap;  
    overflow: hidden;  
    transition: all .2s ease-out;  
  }  
  &__tile {  
    padding: 1.5em;  
    background: #3a2a25;  
    color: white;  
    margin: 0;  
    border: 2px solid gray;  
    border-collapse: collapse;  
  }  
}
```

```
class="game-box__row"  
class="game-box__tile"
```

Rama: i-love-vue-commit-4

```
git add . && git commit -m "4. Matrix using data, and some styles"
```


2.2 Acción para reiniciar la partida

- Creamos un “**eventHub**” para transmitir las acciones

(<https://vuejs.org/v2/guide/migration.html#dispatch-and-broadcast-replaced>)

main.js:

```
Vue.prototype.$eventHub = new Vue();
```

GameInfo:

```
restartGame() {  
  this.$eventHub.$emit('newGame');  
},
```

GameBox:

```
created() {  
  this.$eventHub.$on('newGame', this.resetMatrix);  
},  
destroyed() {  
  this.$eventHub.$off('newGame', this.resetMatrix);  
}
```

Y añadimos el método

```
resetMatrix() {  
  const newMatrix = this.initMatrix(4);  
  this.matrix = newMatrix;  
},
```

Rama: i-love-vue-commit-5

git add . && git commit -m "5. Restart game action using an eventHub"

2.2 Pasamos las acciones al “store”

- GameBox pasa a usar el “store”:

```
import { mapState } from 'vuex';
export default {
  name: 'GameBox',
  computed: {
    ...mapState({
      matrix: (state) => state.matrix,
    }),
  },
};
```

- Store.js

<FUNCION_INIT_MATRIX_AQUI>

```
const GAME_SIZE = 4;
const matrixState = {
  gameSize: GAME_SIZE,
  matrix: initMatrix(GAME_SIZE),
};

export default new Vuex.Store({
  state: matrixState,
  actions: {

  },
  mutations: {

  },
})
```

- Eliminamos el eventHub de “main.js”
- Emitimos la acción “newGame” desde GameInfo:

```
restartGame() {
  this.$store.dispatch('newGame');
},
```

- Añadimos la acción “newGame” al store:

- Acción

```
newGame(context) {
  const newMatrix = initMatrix(GAME_SIZE);
  context.commit('updateMatrix', newMatrix);
}
```

- Mutación

```
updateMatrix(state, matrix) {
  state.matrix = matrix;
}
```

2.3 Creando el patrón aleatorio

- El valor por defecto de las celdas es “empty”
- Antes de devolver la matrix en “initMatrix”, llamamos a una nueva función “fillRandomPattern”

- Rellenamos el patrón aleatorio del tablero de juego

```
const fillRandomPattern = (matrix, patternSize) => {  
  const totalTiles = matrix.length * 2;  
  let leftPatternTiles = patternSize;  
  while (leftPatternTiles > 0) {  
    const newRowRandom = Math.round(Math.random() * (totalTiles - 1));  
    const newColumnRandom = Math.round(Math.random() * (totalTiles - 1));  
    if (newRowRandom >= matrix.length ||  
        newColumnRandom >= matrix.length) {  
      // Avoid going above the matrix boundaries due to rounding  
      continue; // eslint-disable-line no-continue  
    }  
    // Only update the count on cells that are empty  
    const cell = matrix[newRowRandom][newColumnRandom];  
    if (cell.content !== 'pattern') {  
      cell.content = 'pattern';  
      leftPatternTiles -= 1;  
    }  
  }  
};
```

Rama: i-love-vue-commit-6

`git add . && git commit -m "6. Start using the store"`

3. Creando el juego

- Tras generar la nueva matrix de una nueva partida, haremos un delay (setTimeout), tras el cual, ocultaremos las celdas

store.js:

```
// Función "newGame", al final de todo
setTimeout(() => {
  context.dispatch('togglePatternVisibility', false);
}, 2500);

// Nueva "action"
togglePatternVisibility(context, doShow) {
  // Opción 1: usando Array.map
  const updatedMatrix = context.state.matrix.map((row) => {
    return row.map((cell) => Object.assign({}, cell, {display: doShow}));
  });
  context.commit('updateMatrix', updatedMatrix);
  // Opción 2: https://github.com/celdrake/i-love-vue/blob/master/src/store.js#L75
},
```

GameBox:

```
<div class="game-box__tile"
  :class="{ 'is-revealed': cell.display,
           'has-pattern': cell.content === 'pattern' }"
  v-for="(cell, cellIndex) in row" :key="cellIndex">
</div>

// En el bloque "__tile"
&.is-revealed {
  &.has-pattern {
    background-color: blue;
  }
}
```

Rama: i-love-vue-commit-7

`git add . && git commit -m "7. Hiding the pattern after a delay"`

3.1 Creamos un componente para GameTile

- Para poder gestionar mejor las acciones sobre las casillas, las separamos a un componente aparte (GameTile):

GameTile.vue:

```
// Definimos la "prop" "tile"
// Extraemos el bloque SASS de game-box__tile
computed: {
  tileClasses() {
    return {
      'is-revealed': this.tile.display,
      'has-pattern': this.tile.content === 'pattern',
    };
  }
},
```

GameBox.vue:

```
// Importamos el componente "GameTile"
// Lo añadimos en el apartado "Components"
// Modificamos el template y pasamos :tile como propiedad
```

Rama: i-love-vue-commit-8

`git add . && git commit -m "8. GameTile as a separate component"`

3.2 Añadiendo interactividad (I)

- Añadimos el método “tileClick” a la celda.
Hacemos “dispatch” de una acción “revealTile”
- Simplificamos la CSS del componente:

```
tileClasses() {  
  return {  
    'is-revealed': this.tile.display,  
    [`has-${this.tile.content}`]: true,  
  };  
},
```

```
&.is-revealed {  
  &.has-pattern {  
    background-color: blue;  
  }  
  &.has-click-success {  
    background-color: limegreen;  
  }  
  &.has-click-error {  
    background-color: crimson;  
  }  
}
```

3.2 Añadiendo interactividad (II)

- En el store, añadimos las acciones asociadas

// Añadimos nuevas propiedades al estado

```
const gameState = {  
  gameSize: GAME_SIZE,  
  revealedTiles: 0,  
  successTiles: 0,  
  matrix: initMatrix(GAME_SIZE),  
};
```

// Añadimos la acción que gestiona el click de una casilla

```
revealTile(context, tile) {  
  const state = context.state;  
  if (state.revealedTiles === state.gameSize) {  
    return;  
  }  
  const isSuccess = tile.content === 'pattern';  
  const totalRevealed = state.revealedTiles + 1;  
  // We mutate the state directly, so not using an action here  
  tile.content = isSuccess ? 'click-success' : 'click-error';  
  tile.display = true;  
  context.commit('setRevealedTiles', {  
    revealed: totalRevealed,  
    success: state.successTiles + (isSuccess ? 1 : 0)  
  });  
},
```

// Definimos la mutación para el cálculo del resultado de la partida

```
setRevealedTiles(state, result) {  
  state.revealedTiles = result.revealed;  
  state.successTiles = result.success;  
},
```

// Recordar lanzar esta mutación cuando comienza una nueva partida!!

Rama: i-love-vue-commit-9

`git add . && git commit -m "9. Tile can be clicked. Status change"`

3.3 Resultado final del juego (I)

// Detectamos el final del juego y lanzamos la acción “onEndGame”

```
onEndGame(context) {  
  context.state.matrix.forEach((row, rowIndex) => {  
    row.forEach((cell, cellIndex) => {  
      const cellCount = (rowIndex * GAME_SIZE) + cellIndex;  
      // Podemos mutar directamente (o usar “updateTile”)  
      setTimeout(() => {  
        cell.display = true;  
        // TODO definir propiedad a nivel de “matrix”  
        cell.showResult = true;  
      }, 200 * cellCount);  
    });  
  });  
},  
},
```

// Recordar poner “showResult” a false inicialmente

// Mutación para modificar una celda:

```
updateTile(state, { row, column, updatedTile }) {  
  const updatedMatrix = Object.assign({}, state.matrix);  
  updatedMatrix[row][column] = updatedTile;  
  
  state.matrix = updatedMatrix;  
},
```

Rama: i-love-vue-commit-10

git add . && git commit -m "10. Showing end game result"

3.3 Resultado final del juego (II)

// Añadimos el símbolo que indica el final del juego

```
<span class="game-tile__symbol"
      :class="{ 'show-result': tile.showResult }">
  {{ endResult }}
</span>
```

// Añadimos la propiedad computada

```
endResult() {
  switch(this.tile.content) {
    case 'click-success':
      return '😄';
    case 'pattern':
      return '🧠';
    case 'click-error':
      return '🙄';
    case 'empty':
    default:
      // Return a symbol even for empty cells,
      // to keep the same spacing
      return '👉';
  }
},
```

Si no se ven los emojis, clicar aquí: [Link emojis](#)

// Añadimos la nueva CSS de “game-tile”:

```
&__symbol {
  font-size: 1.5em;
  visibility: hidden;
  &.show-result {
    visibility: visible;
  }
}
```

// Ponemos 4em de margin a la “game-tile”