TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE:

- Mutations.
- Getters.
- MapGetters.

MUTATIONS:

La única forma en la que podemos cambiar estados (states) en Vuex, es a través de una mutación. Las mutaciones en Vuex son muy similares a los eventos: cada una tiene un tipo (string) y un handler (controlador), que es donde se hará la modificación del estado. Éste recibe **state** como primer argumento.

```
const store = new Vuex.Store({
  state: {
    count: 1
  },
  mutations: {
    increment (state) {
      // mutate state
      state.count++
    }
  }
})
```

Si observamos, nos daremos cuenta de que tenemos un state llamado “count”, el cual es un número. Si queremos cambiar su valor, debemos crear una mutación, la cual se encargará de realizar dicho cambio.

En el ejemplo se crea una mutación llamada “increment”, que recibe como primer argumento a state, el cual tiene acceso a todos, y nos permite modificarlos. En este caso, por cada vez que se llama a la mutación, el valor de count se incrementará en 1.

No podemos llamar a una mutación directamente. Debemos pensar que es similar a un evento: Cuando una mutación llamada “increment” sea gatillada, se ejecutará la lógica del controlador. Para invocar/gatillar la mutación, usaremos: “**store.commit**”.

```
store.commit('increment')
```

COMMIT CON ARGUMENTO:

A veces, es necesario entregar algún dato a la mutación, para que ésta pueda generar la lógica y cambiar en el estado. Es por ello, que podemos pasar un argumento adicional a “store.commit”, el cual es llamado payload.

```
// ...
mutations: {
  increment (state, n) {
    state.count += n
  }
}
```

```
store.commit('increment', 10)
```

Si necesitarnos pasar más de un dato, deberíamos utilizar un objeto como payload.

GETTERS:

A veces podríamos necesitar computar estados, basados en datos del state de Vuex. Por ejemplo: la forma que vimos de poder hacer esto, sería en una propiedad computada nueva, en la que tomaríamos el dato de Vuex, y podríamos generar lógica.

```
computed: {
  doneTodosCount () {
    return this.$store.state.todos.filter(todo => todo.done).length
  }
}
```

Si más de un componente necesitara hacer uso de esto, tendríamos que duplicar la función de cada uno.

Vuex nos permite definir “getters” en el store. Podemos pensar en ellos, como las propiedades computadas para el store de Vuex.

Los getters reciben “state” como 1er argumento.

```
const store = new Vuex.Store({
  state: {
    todos: [
      { id: 1, text: '...', done: true },
      { id: 2, text: '...', done: false }
    ]
  },
  getters: {
    doneTodos: state => {
      return state.todos.filter(todo => todo.done)
    }
  }
})
```

ACCEDER A LOS GETTERS:

Se hace desde un componente.

```
store.getters.doneTodos // -> [{ id: 1, text: '...', done: true }]
```

MAPGETTERS

Al igual que los estados pueden ser invocados por mapState, los getters pueden ser invocados por mapGetters.

```
import { mapGetters } from 'vuex'

export default {
  // ...
  computed: {
    // mix the getters into computed with object spread operator
    ...mapGetters([
      'doneTodosCount',
      'anotherGetter',
      // ...
    ])
  }
}
```