

State Management



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://about.softuni.bg>

Table of Contents

1. State management
2. Pinia
3. Helpers



Have a Question?



sli.do

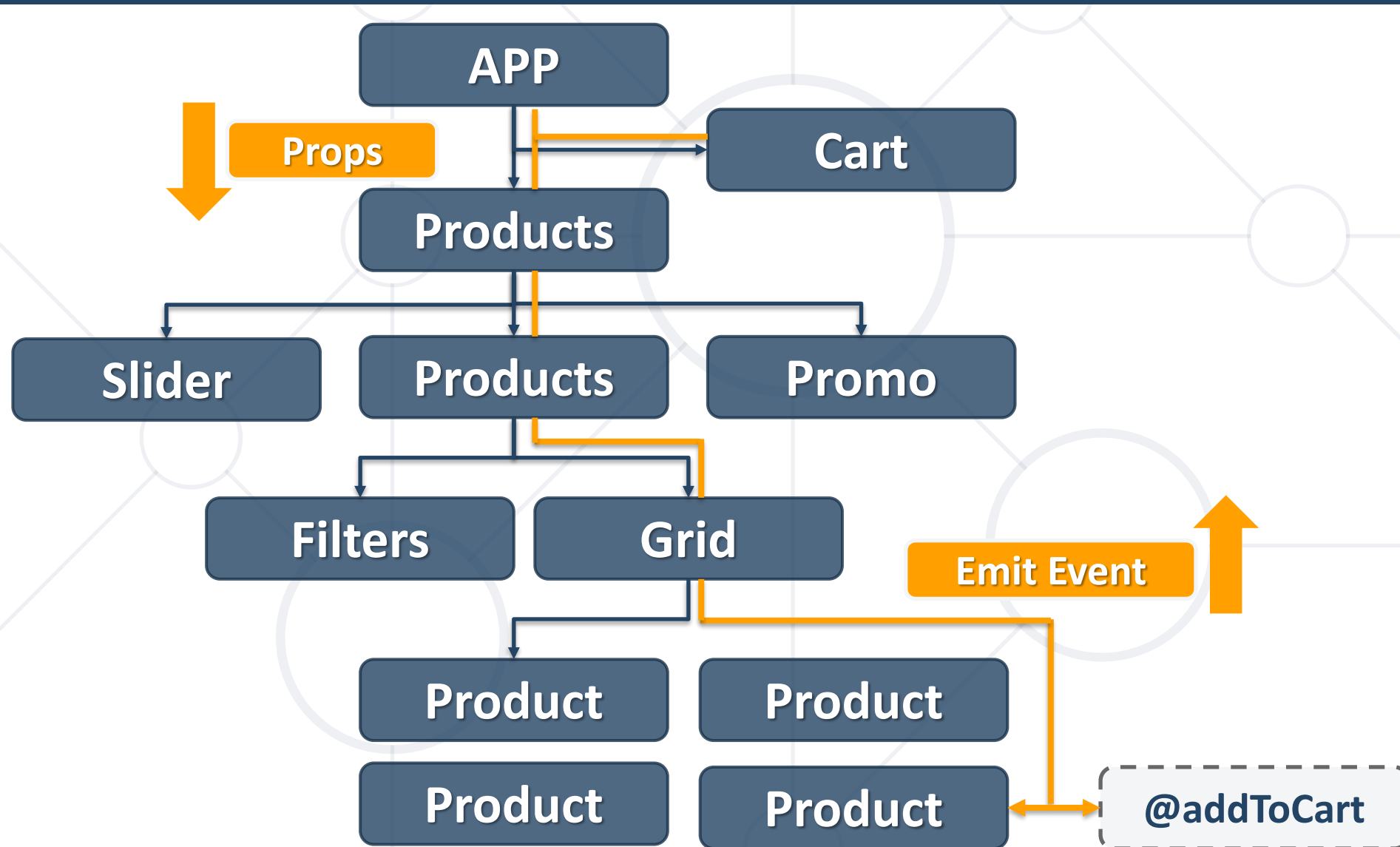
#vue-js



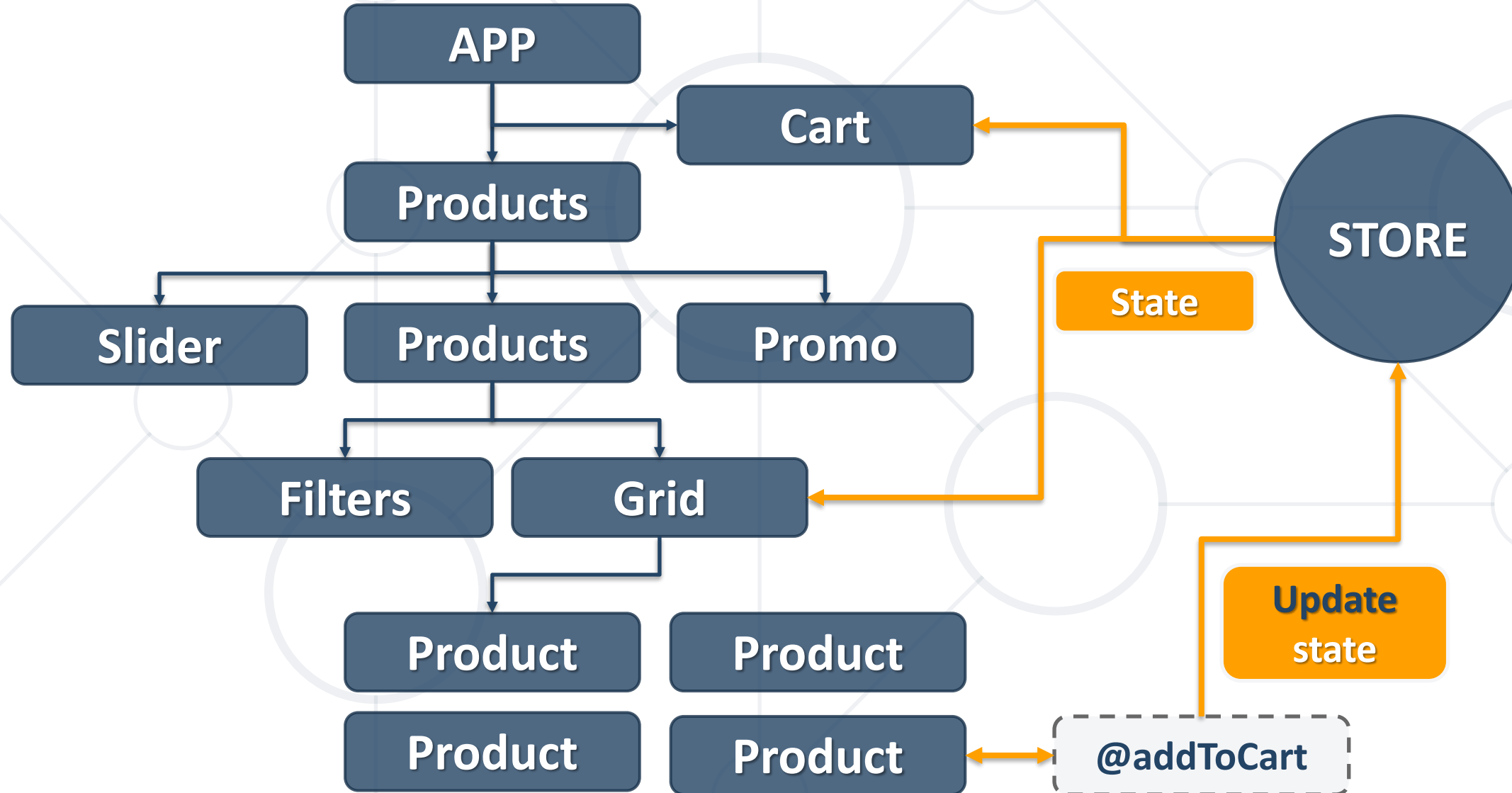
State Management

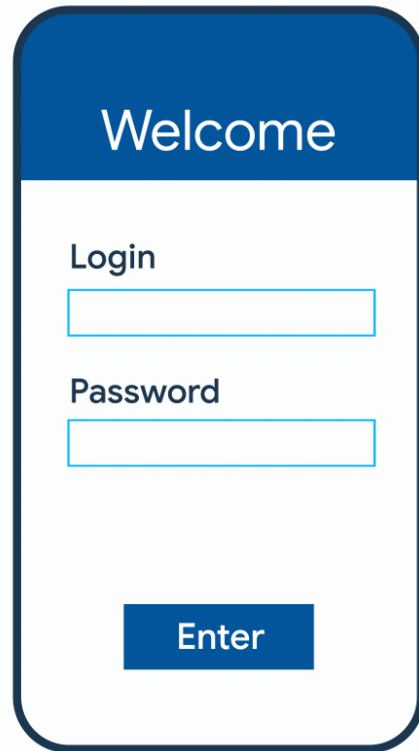
Why Do We Need It?

The Problem: Communication

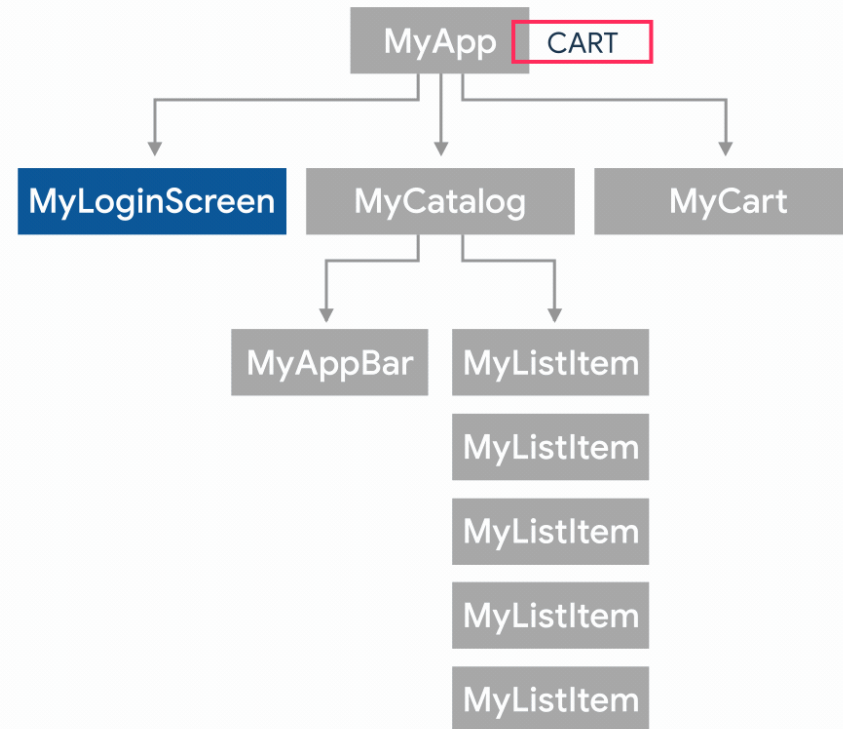


The Solution: Centralized State





A mobile application interface mockup. It features a blue header with the text "Welcome". Below the header, there is a "Login" section with two input fields: one for the username and one for the password. At the bottom of the login section is a blue button labeled "Enter".



Provide and Inject

Parent

```
export default {  
  data() {  
    return {  
      message: 'hello!'  
    }  
  },  
  provide() {  
    return {  
      message: this.message  
    }  
  }  
}
```

Child

```
export default {  
  inject: ['message'],  
  data() {  
    return {  
      // initial data based on  
      injected value  
      fullMessage: this.message  
    }  
  }  
}
```


When should I use a Store

- Data that is used in many places
- A very complicated multi-step form
- Avoid including in the store local data
 - The visibility of an element local to a page
 - Variable specific to the current component only





Pinia

State Management Pattern for Vue

- Install via npm
- Register as plugin

```
npm install pinia
```

```
import { createApp } from 'vue'  
import { createPinia } from 'pinia'  
import App from './App.vue'  
  
const pinia = createPinia()  
const app = createApp(App)  
  
app.use(pinia)  
app.mount('#app')
```

Defining a Store

```
// stores/counter.js
import { defineStore } from "pinia";

export const useCountStore = defineStore("count", {
  state: () => ({ count: 0 }),
  getters: {
    doubledCount: (state) => state.count * 2,
  },
  actions: {
    increment() {
      this.count++;
    },
  },
});
```



```
<script>
import { useCountStore } from "../stores/counter";
export default {
  setup() {
    const countStore = useCountStore();
    return { countStore };
  },
};
</script>
```

```
<template>
  <h2>{{ countStore.doubledCount }}</h2>
  <button @click="countStore.increment()">Increment</button>
</template>
```

Mutating the state

- By default, you can directly read and write to the state by accessing it through the store instance

Mutating directly

```
<script>
  methods: {
    addFive() {
      this.countStore.count += 5;
    },
  },
};
</script>
```

Multiple state updates

```
countStore.$patch((state) => {
  state.items.push({ name: 'shoes',
    quantity: 1 })
  state.hasChanged = true
})
```

```
countStore.$patch({
  count: store.count + 1,
  name: 'DIO',
})
```

Reset changes

```
<template>
  <button
    @click="countStore.$reset()">Reset</button>
</template>
```

- Equivalent of computed values for the state of a Store
- Defined with the **getters** property in **defineStore()**
- Receive the **state** as the **first parameter** to encourage the usage of arrow function

```
export const useCounterStore = defineStore('counter', {  
  state: () => ({  
    count: 0,  
  }),  
  getters: {  
    doubledCount: (state) => state.count * 2,  
    doubledCountPlusOne() {  
      return this.doubledCount + 1  
    },  
  },  
})
```



- Equivalent of methods
- Defined with the **actions** property in `defineStore()`
- Perfect to define business logic

```
export const useCountStore = defineStore("count", {
  state: () => ({ count: 1, randomNumber: 0 }),
  getters: {
    doubledCount: (state) => state.count * 2,
  },
  actions: {
    increment() {
      this.count++;
    },
    random() {
      this.randomNumber = this.count +
Math.random();
      // Or a getter
      this.randomNumber = this.doubledCount +
Math.random();
    },
  },
});
```


Options API map helpers

```
<template>
  <h2>Access the store state as a computed: {{ count }}</h2>
</template>

<script>
  import { useCountStore } from "../stores/counter";
  import { mapState, mapActions } from "pinia";

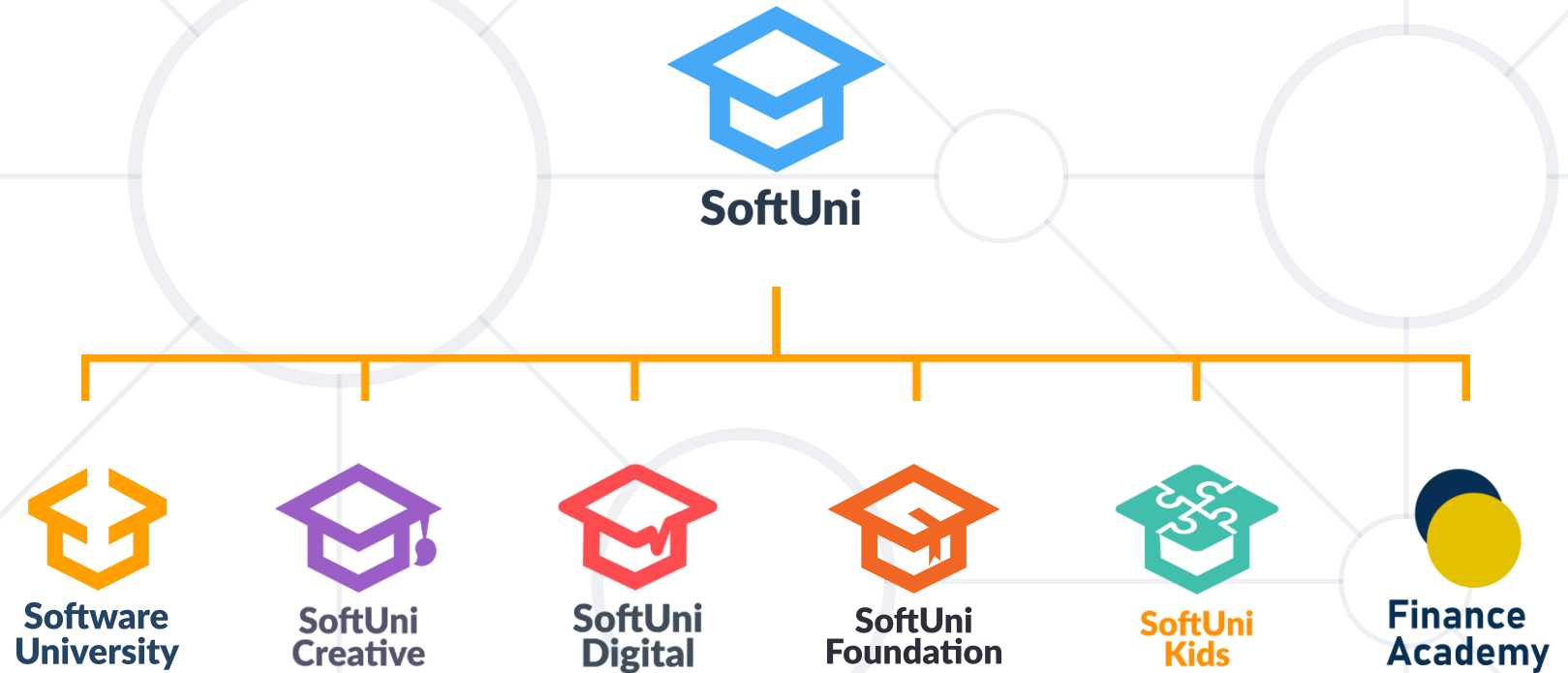
  export default {
    computed: {
      ...mapState(useCountStore, ["count", "doubledCount"]),
    },
    methods: {
      ...mapActions(useCountStore, ['increment']),
    },
  };
</script>
```



- State management libraries help us write easier and maintainable state and updates
- Pinia is an official plugin to add extended features for Vue
- Pinia stores are equivalent to a component's Options like state, getters and actions



Questions?



SoftUni Diamond Partners

**SUPER
HOSTING
.BG**



**Coca-Cola HBC
Bulgaria**

 **Flutter**TM
International

INDEAVR
Serving the high achievers



AMBITIONED

 **DRAFT
KINGS**



**SOFTWARE
GROUP**



BOSCH



Postbank

Решения за твоето утре

 **PHAR
VISION**



SmartIT

DXC
TECHNOLOGY

createX

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg

