STU FIIT
SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY
A INFORMAČNÝCH TECHNOLÓGIÍ

Metodika pre prácu s Frontend časťou aplikácie
DUtí(m)

Tím číslo: 18
Členovia tímu: Bende Tomáš, Búcsiová Veronika, Horniczká Veronika, Melicherčík Jozef, Pacher Marek, Pavkovček Filip, Slíž Boris, Sojka Michal
Akademický rok: 2018/2019
Dátum vypracovania: 15.10.2018
Dátum poslednej úpravy: -
Vypracoval: Michal Sojka

1. Úvod

Účelom tejto metodiky je udržiavanie jednotných štandardov a postupov pri vyvíjaní frontendovej časti projektu. Táto metodika opisuje konvencie písania zdrojového kódu a komentárov, ktoré by mali členovia tímu dodržiavať pri rozširovaní, alebo vytváraní funkcionality, ktorá sa týka frontendovej časti aplikácie.

Nakoľko je aplikácia vyvíjaná vo frameworku Laravel, ktorý zväzuje obe časti (frontend a backend), rozhodli sme sa, že nebudeme striktne udržiavať rovnaké konvencie kódu pre backend a frontend časť. Obidve časti budú pokrývať samostatné metodiky, aj keď nie je vylúčené, že v budúcnosti sa tieto metodiky budú zlúčené do jednej.

Rozdielnymi metodikami chceme pomôcť zabezpečiť aspoň čiastočné rozlíšenie týchto dvoch častí v projekte.

2. Konvencie písanie zdrojového kódu

Pri písaní nového zdrojového kódu sa vždy zameriavame na čitateľnosť a prehľadnosť kódu. Pri projekte, kde väčšina členov ešte nemá skúsenosti s použitým frameworkom takto zabezpečíme lepšiu efektivitu a rýchlosť vývoja.

1. Názvy premenných

Premenné použité pri renderovaní, alebo pomocných funkcionalít sa snažíme pomenovávať výstižne a presne. Takto bude zdrojový kód lepšie čitateľný a ostatní členovia tímu ho rýchlejšie pochopia.

Pri pomenovaní premenných používame camelCase formát zápisu.

Nesprávne	Správne
$a = 1;	$activityCount = 1;
$temp = $array[i];	$targetFunctionality = filteredActivity[index];
$i	$index
public doStuff(int i, User array) {...}	public filterUsers(int index, User UserArray) {...}

2. **Opakovanie a množstvo kódu**

Pri písaní nového kódu sa snažíme dodržiavať princíp "*Do not repeat yourself."* Preto, ak sa niekde opakuje **3 a viac** rovnakých riadkov kódu, snažíme sa ich vyňať do osobitnej metódy, alebo modulu. Tiež sa snažíme funkcie vhodne parametrizovať, aby sme ich mohli používať univerzálne.

Nesprávne

```
public translateActivityTableTargetRow(int index, Table activityTable) {
  $element = document.getElementById($activityTable.id);
  $rows = $element.target.rows;
  $targetRow = $rows[index] || -1;

  $translatedRow = translator.translate($targetRow);

  return $translatedRow;
}


public modifyEventsTableTargetRow(int index, Table eventsTable) {
  $element = document.getElementById(eventsTable.id);
  $rows = $element.target.rows;
  $targetRow = $rows[index] || -1;

  $modifiedTargetRow = $targetRow + 'abc';

  return $modifiedTargetRow;
}
```

Správne

```
private getTargetElementRow(int rowIndex, Table targetTable) {
  $ element = document.getElementById(targetTable.id);
  $rows = $element.target.rows;
  $targetRow = $rows[index] || -1;

  retrun $targetRow;
}
```

```
public translateActivityTableTargetRow(int index, Table activityTable) {
  $targetRow = getTargetElementRow(index, activityTable);

  $translatedRow = translator.translate($targetRow);

  return $translatedRow;
}

public modifyEventsTableTargetRow(int index, Table eventsTable) {
  $targetRow = getTargetElementRow(index, eventsTable)
  $modifiedTargetRow = $targetRow + 'abc';

  return $modifiedTargetRow;
}
```

3. CSS triedy

Pri pomenovaní CSS tried dodržiavame **snake-case** syntax. Jednotlivé vlastnosti v triede vždy radíme **abecedne**. Triedy zoraďujeme podľa veľkosti elementov, ktoré používajú danú CSS kolekciu tried. Typicky je to page-container > section-container > collection-container > collection-item. Media queries píšeme **pod** definované triedy.

Nesprávne

```
@media only screen and (max-width: 576px) {
  .navigation-item {
    flex-grow: 1;
    flex-basis: 0;
    display: flex;
    text-align: center;
  }
  .navigation-container {
    display: none;
    animation: rotate 3.5s fade-in;
  }
}

.navigation-container {
  display: flex;
  width: 100%;
  align-items: center;
  border: 1px solid black;
```

```
  justify-content: center;
}
```

Správne

```
.navigation-container {
  align-items: center;
  border: 1px solid black;
  display: flex;
  justify-content: center;
  width: 100%;
}

@media only screen and (max-width: 576px) {
  .navigation-container {
    animation: rotate 3.5s fade-in;
    display: none;
  }

  .navigation-item {
    display: flex;
    flex-grow: 1;
    flex-basis: 0;
    text-align: center;
  }
}
```

3. Konvencie písanie komentárov

Pre správne využitie *phpDocumentator* je potrebné dodržiavať správne konvencie písania komentárov.

Jedno riadkový *DocComment* napíšeme ako:

```
/** This is a single line DocComment. */
```

Viac riadkový *DocComment* napíšeme ako:

```
/**
* This is a multi-line DocComment.
*/
```

Komentáre píšeme vždy **pred** časť kódu, ktorého sa týkajú. Komentáre sa snažíme písať v tých prípadoch, keď potrebujeme upozorniť na vlastnosti kódu, ktoré nie sú hneď samozrejmé z pohľadu na zdrojový kód. Komentáre **nepíšeme** v prípadoch keď sa snažíme vysvetliť čo daný kód **vykonáva**, ale keď sa snažíme vysvetliť, **prečo** sme zvolili konkrétny postup. Takisto nekomentujeme len jednotlivé riadky.

Nesprávne

```
public createGenerator() { ... }
/**
* This function instantiates new generator and initializes starting value.
* After initialization of the starting value the generator is automatically
* called at the execution point of invocation.
*/
```

Správne

```
/**
* We used dynamic way of creating generator, so we can use them as public
* methods anywhere in execution of project. Starting value initialization
isused
* for safety reason and fault-proof execution.
*/
public createGenerator() { ... }
```

4. Konvencie pre React.js

Pri používaní knižnice React pri práci na frontende je dôležité uplatňovať niektoré všeobecné princípy. Táto časť sa nezaoberá základmi písania React, slúži na uplatňovanie písania vhodnej štruktúry zdrojového kódu.

1. **Stateless vs stateful component**
   Rozdiel medzi stateless a stateful komponenty je absencia vlastného stavu.

   Stateful komponent sa vytvára v prípadoch, keď v danom komponente potrebuje udržiavať vlastný stav a nie je možné využiť *props*. V takomto prípade vytvárame komponent ako triedu a priraďujeme mu vlastný stav. Stateful komponenty väčšinou zodpovedajú za nejakú časť biznis funkcionality a odosielajú dáta do nižších komponentov na základe hierarchickej štruktúry.

   Stateless komponenta sa vytvára v prípade, že nevieme ďalej rozumne rozdeliť väčšie komponenty a jediné čo potrebujeme je zobrazenie dát. Tento prípad väčšinou nastáva keď potrebujeme zobraziť len HTML markup a nepotrebujeme použiť JSX. Stateless komponenty nedisponujú vlastným stavom a dáta zobrazujú výlučne na základe props.

2. **Function binding**
   Laravel React preset prichádza z vopred definovanou webpack konfiguráciou a taktiež automatickým transpilingom EcmaScript štýlu zápisu zdrojového kódu. Táto pôvodná konfigurácia zostáva nemenná, preto sa pri písaní React komponentov **nemôžu** používať **arrow functions**.

   Namiesto týchto funkcií využívame function binding v konštruktore triedy, v ktorej sú vytvorené.

   Príklad zápisu:

```
constructor(props) {
        super(props);
        this.handleInput = this.handleInput.bind(this);
}
```

3. **Code splitting**

   Základným princípom React-u je hierarchické rozdelenie na znovupoužitelné komponenty. V projekte používame nasledovné rozdelenie: Page-container > Section-container > Collection-container > Collection-item. Dobrým indikátorom dodržiavania tohto rozdelenia je to, že posledné logické komponenty slúžia len na zobrazenie dát, ktoré prijímajú cez *props*.

   Na tento princíp sa dbá najmä pri code review pri funkcionalite, ktorá je implementovaná pomocou React-u. Môžu existovať špeciálne prípady, kedy takéto rozdelenie nie je možné dodržať. Takéto prípady musia byť schválene hlavným architektom a hlavným frontend developerom po vzájomnej konzultácií.