

# Metodika pre prácu s Frontend časťou aplikácie DUtí(m)

**Tím číslo:** 18

**Členovia tímu:** Bende Tomáš, Búcsiová Veronika, Horniczka 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 striktné 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 a 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
<code>\$a = 1;</code>	<code>\$activityCount = 1;</code>
<code>\$temp = \$array[i];</code>	<code>\$targetFunctionality = filteredActivity[index];</code>
<code>\$i</code>	<code>\$index</code>
<code>public doStuff(int i, User array) {...}</code>	<code>public filterUsers(int index, User UserArray) {...}</code>

## 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;  
  
    return $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 zoradíme 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 a 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 funtion 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 is used  
 * for safety reason and fault-proof execution.  
 */  
public createGenerator() { ... }
```