Dokumentacja techniczna komponentów interaktywnych

Wprowadzenie

Dokumentacja opisuje techniczne zagadnienia związane z tworzeniem własnych komponentów interaktywnych.

Przed utworzeniem własnego komponentu należy zarezerwować nazwę w systemie. Nazwa składa się z dwóch członów oddzielonych znakiem slash: nazwa przestrzeni/kod silnika (np. core/geogebra).

Po zarezerwowaniu nazwy, system utworzy repozytorium GIT do którego należy wgrać potrzebne pliki. System rozpocznie budowanie silnika, po otrzymaniu commita.

Wymagania techniczne

Kod silnika musi być zgodny z ECMAScript5. Wszystkie pliki tekstowe muszą być zakodowane w UTF-8 bez sygnatury BOM.

W dokumentacji użyto słowa kluczowego **async** (jest częścią ECMAScript6). Oznacza ono że dana metoda może zwrócić obietnicę Promise (nie musi).
Jeśli przy tworzeniu komponentu, używane są nowoczesne struktury składniowe, kod **musi** zostać skompilowany do składni ECMAScript5 (np. przy użyciu komilatora babeljs).

Architektura silnika

Silnik jest aplikacją zgodną z interfejsem platformy. Aby uruchomić silnik, należy utowrzyć instancję.

Musi zawierać plik engine.json oraz plik wejścia.

Definicja silnika - engine.json

Plik engine.json definiuje jakie funkcje dostarcza oraz jakich funkcji wymaga komponent.

Włączenie niektórych opcji, może nieść za sobą konieczność implementacji dodatkowych metod.

Minimalna (wymagana) zawartość pliku:

```
{
  "entry": "entry.js",
}
```

Wszystkie opcje

```
{
  "entry": "entry.js",
  "stateful": true | false,
  "isolation": "shadow" | "iframe" | "none",
  "validation": "auto" | "manual" | "none",
  "editor": {
    "entry": "editorEntry.js",
    "defaultData": {},
    "demoData": {}
  }
}
```

Funkcje poszczególnych pól:

- **entry**: określa plik wejścia
- **isolation**: sposób izolacji jaki zapewnia silnik.

  shadow (Ǩŕ ŅăŷłŐǴ) - komponent zostanie zainicjalizowany wewnątrz Shadow DOM.
  Jeśli przeglądarka nie obsługuje tej metody, użyty będzie iframe.

  iframe - system tworzy ramkę iframe.

  none - brak izolacji. Dozwolone tylko gdy silnik sam zapewnia sobie izolację (np umieszcza iframe).

- **data**: dane które przekazywane są potem w metodzie init
- **stateful**: określa czy aplikacja zapisuje stan.

  Szczegóły implementacyjne zostały opisane w sekcji stateful

- **validation**: określa czy ćwiczenie może podlegać ocenie.

  auto - komponent sam sprawdza poprawność wykonania. Wymaga implementacji dodatkowych metod.
  Szczegóły implementacyjne zostały opisane w sekcji validation

  manual - poprawność wykonania sprawdzana jest manualnie, przez użytkownika systemu

  none (Ǩŕ ŅăŷłŐǴ) - poprawność wykonania nie podlega ocenie

- **editor**: określa parametry umożliwiające utworzenie edytora komponentu
  - **entry**: określa plik wejścia dla modułu edytora
  - **defaultData**: domyślne dane, przy tworzeniu nowego komponentu
  - **demoData**: dane wykorzystywane przy użyciu opcji "Wczytaj dane demonstracyjne"

Punkt wejścia (entry)

Dostarczany jest w formie asynchronicznego modułu AMD lub kompatybilnego (np UMD). Synchroniczne żądanie bibliotek jest niedozwolone.

Moduł musi eksportować fabrykę (funkcję która utworzy silnik) lub konstruktor.

Przykłady eksportu fabryki (moduł AMD):

```
define([], function() {
    return function() {
        return {
            init: function() {},
            destroy: function() {}
        }
    }
})
```

Przykłady eksportu konstruktora (moduł AMD):

```
define([], function() {
    function Engine() {}

    Engine.prototype.init: function() {},
    Engine.prototype.destroy: function() {}

    return Engine;
})
```

Przykłady eksportu fabryki (moduł AMD) z żądaniem biblioteki jQuery:

```
define(['jquery'], function($) {
    return function() {
        return {
            init: function() {},
            destroy: function() {}
        }
    }
})
```

Biblioteki współdzielone

- vue (wersja 2)
- jquery (wersja 2)
- underscore
- backbone
- axios
- react (wersja 16)

Jeśli istnieje potrzeba użycia innych popularnym frameworków, prosimy o kontakt z administracją portalu.

Interfejs silnika

Bazowy interfejs silnika. Implementacja tego interfejsu jest zawsze wymagana.

```
{
    async init(container, api, options)
    destroy(container)
}
```

- **async init(container, api, options)**

  Metoda inicjalizująca komponent

  - **container** - obiekt DOM w którym nalezy umieścić komponent.
  - **api** - obiekt API pozwalający na komunikację z systemem
  - **options** - opcje dodatkowe:

    ```
    {
        contrastMode: false | "yellowOnBlack" | "blackOnYellow" | "whiteOnBlack",
        locale: "pl_PL",
        showAnswers: true | false,
        data: {...}
    }
    ```

    W polu data, przekazane są wartości z pliku manifest.json instancji.

    Uwaga: dane przekazane w tym polu powinny być traktowane jako niebezpieczne/nieprzefiltrowane. Ich poprawne wykorzystanie leży po stronie silnika (patrz bezpieczeństwo).

Obiekt zwracając obietnice, sygnalizuje że nie zakończył jeszcze inicjalizacji. System wyświetli informację o ładowaniu do czasu ukończenia obietnicy. W przypadku wystąpienia błędu wyświetlony zostanie systemowy komunikat o braku możliwości uruchomienia komponentu.

Przykład ładowania css:

```
init(container, api, options) {
    return api.loadCss(
        api.enginePath('dist/entry.css')
    ).then(function() {
        console.log('CSS został załadowany');
    });
}
```

- **destroy(container)**

  Metoda która jest wykonywana przed zniszczeniem komponentu.
  Jej zadaniem jest usunięcie wszystkich elementów DOM, zdarzeń i uwolnienie wszystkich innych zasobów które są używane przez komponent.

Stateful

Po zmiane stanu, komponent powinien wywołać metodę API **triggerStateSave()**. System wkrótce wywoła metodę **getState()**. Zwrócona wartość zostanie zapisana.

```
{
    setState(stateData)
    getState()
    setStateFrozen(isFrozen)
}
```

- **setState(stateData)** - przywraca stan komponentu. Przy pierwszym uruchomieniu stateData będzie wartością NULL.

  Uwaga: Dane zawarte w stateData powinny być traktowane jako niebezpieczne. Ich poprawne wykorzystanie leży po stronie silnika (patrz bezpieczeństwo).

- **getState()** - Zwraca aktualny stan komponentu. Wartość musi być serializowalna

- **setStateFrozen(isFrozen)** - Ustawia stan zamrożenia. Metoda wywoływana jest zawsze po metodzie setState.

  Po wywołaniu setStateFrozen(true), komponent musi być zablokowany. Nie może wtedy zmieniać stanu.

  Po wywołaniu setStateFrozen(false), komponent musi powrócić do normalnego trybu pracy.

Validation

Wymaga implementacji interfejsu stateful.

```
{
    isStateValid(stateData)
}
```

- **isStateValid(stateData)** - weryfikuje poprawność wykonania, na podstawie danych dostarczonych w argumencie. Zwraca wartość logiczną

API

Obiekt API jest przekazywany w metodzie init. Umożliwia on komunikację z systemem.

```
{
    triggerStateSave();
    triggerStateRestore();
    enginePath(path);
    dataPath(path);
    async loadCss(realPath);
}
```

- **triggerStateSave()** - metoda powinna zostać wywołana po zmianie stanu kompoentu

- **triggerStateRestore()** - dodatkowa metoda, która wymusza przywrócenie stanu komponentu z bazy danych. Używanie jej nie jest konieczne.
- **enginePath(path)** - zwraca ścieżkę relatywną względem katalogu silnika
- **dataPath(path)** - zwraca ścieżkę relatywną względem katalogu danych
- **async loadCss(realPath)** - ładuje plik CSS

Instancja komponentu

Gotowy silnik można wykorzystać, tworząc instancję komponentu.

Komponent składa się z pliku manifest.json oraz innych plików z których korzysta silnik, a które są specyficzne dla danej instancji.

Struktura pliku **manifest.json**

```
{
    "engine": "nazwa przestrzeni/kod silnika",
    "data": {...}
}
```

Wszystkie pliki powinny zostać spakowane do formatu ZIP. Plik manifest.json musi znajdować się na samym szycie struktury archiwum (nie może znajdować się w żadnym katalogu).

Bezpieczeństwo

Warunkiem koniecznym jest, aby silnik nie pozwalał na wykonanie obcego kodu lub zpreparowanych żądań.

Jeśli silnik ma zostać udostępniony osobom trzecim, konieczne jest aby dodatkowo filtrował wszystkie dane przekazane w pliku manifest.json.

aktualizacja: 1 Czerwca 2020