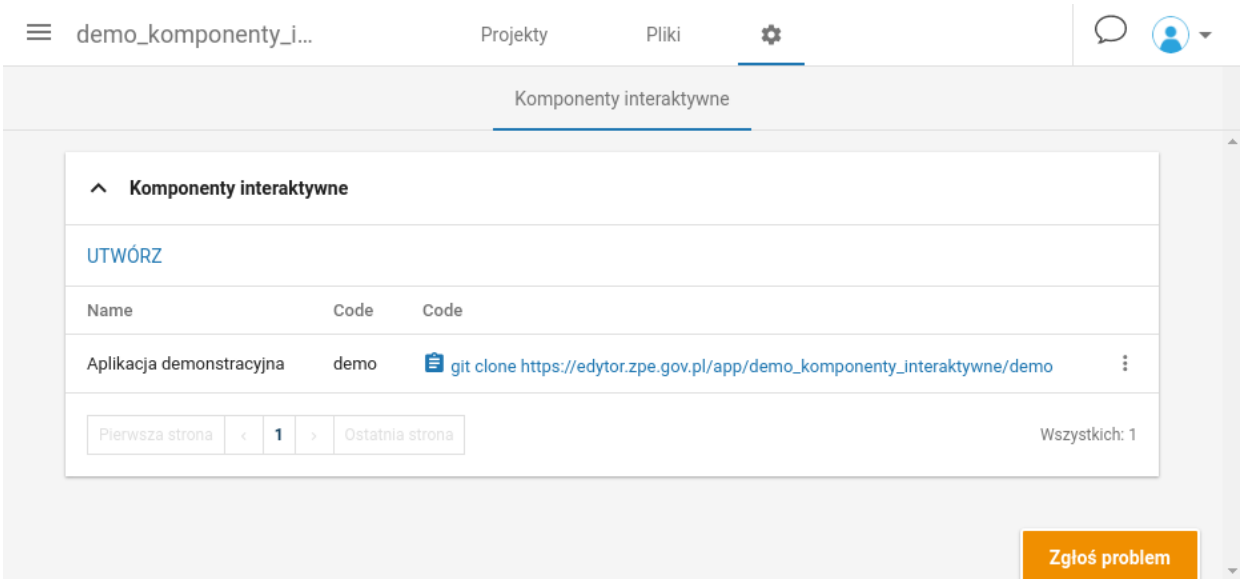
Dokumentacja techniczna komponentów interaktywnych

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 zminifikowane źródła aplikacji oraz inne potrzebne pliki (obrazy, CSS które są używane zawsze).

Pliki które dotyczą konkretnej instancji ćwiczenia, powinny być dostarczone z danymi.

Wymagania techniczne

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

Jeśli przy tworzeniu komponentu, używane są nowoczesne struktury składniowe, kod **musi** zostać skompilowany do składni ECMAScript5 (np. przy użyciu kompilatora babeljs).

Obiekty interaktywne muszą zapewniać responsywność i kompatybilność z:

- Firefox (ostatnia wersja -1)
- Chrome (ostatnia wersja -1)
- Opera (ostatnia wersja -1)
- Microsoft Edge 16
- Safari 12+
- iOS Safari 12+

Architektura silnika

Silnik jest aplikacją zgodną z interfejsem platformy. Wszystkie pliki silnika umieszczone są w repozytorium GIT utworzonym przez system.

Na silnik składają się:

- engine.json **(wymagany)**
- punkt wejścia **(wymagany)**, jego położenie jest określone w engine.json
- punkt wejścia edytora *(opcjonalny)*, jego położenie jest określone w engine.json
- inne pliki statyczne *(opcjonalnie)*

Aplikacja może odnosić się **tylko** do plików znajdujących się w repozytorium (używając metody enginePath) lub w konkretnej instacji (używając metody dataPath).

Aby uruchomić silnik, należy utworzyć instancję.

Silnik powinien zawierać tylko kluczowe pliki konieczne do uruchomienia widżetu. Zbyt duży silnik może utrudnić pobieranie materiałów do trybu offline.

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,
  "printable": true | false,
  "validation": "auto" | "manual" | "none",
  "useWebGL": true | false,
  "editor": {
    "entry": "editorEntry.js",
    "defaultData": {},
    "demoData": {}
  }
}
```

Funkcje poszczególnych pól:

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

  Wymaga implementacji dodatkowych metod. Szczegóły implementacyjne zostały opisane w sekcji stateful.
- **printable**: aplikację można drukować (wraz z nim dostarczone są style CSS do druku).
- **useWebGL**: deklaracja czy komponent używa WebGL.
- **validation**: określa czy ćwiczenie może podlegać ocenie.

  Możliwe wartości:

  - auto - komponent sam sprawdza poprawność wykonania.
    Wymaga implementacji interfejsu validation.
  - manual - poprawność wykonania sprawdzana jest manualnie, przez użytkownika systemu.
    Wymaga implementacji interfejsu stateful.
  - none (*domyślnie*) - 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).

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() {}
        }
    }
})
```

Przykładowy fragment konfiguracji webpack pozwalający na eksport biblioteki:

```
module.exports = {
    // ...
    output: {
        libraryTarget: 'amd'
    },
    externals: {
        jquery: 'jquery'
    }
};
```

Przykładowy punkt wejścia dla powyższej konfiguracji:

```
import $ from 'jquery'

function Engine() {}

Engine.prototype.init = function() {}
Engine.prototype.destroy = function() {}

export default Engine;
```

Używanie bibliotek

Razem z systemem dostarczone są popularne biblioteki, które można wykorzystać do tworzenia widżetów. Używając załadowaną już bibliotekę, można mocno ograniczyć rozmiar tworzonych komponentów.

Wśród nich można znaleźć:

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

Bibliotek tych, nie należy modyfikować (np. dodawać pluginów).

Jeśli istnieje potrzeba użycia innych popularnym frameworków, które warto dostarczyć w systemie, prosimy o kontakt z administracją portalu.

Dostawca może użyć dowolnego otwartego oprogramowanie i dostarczyć go razem z komponentem.

Kryteria jakościowe

Warunki, jakie musi spełnić kod silnika (kod komponentu oraz całego dostarczonego z nim oprogramowania)

- nie można modyfikować obiektów DOM poza kontenerem dostarczonym w metodzie `init()`.
- nie można modyfikować zmiennych publicznie (w szczególności, w obiekcie `window`).
- nie można dostarczać rozwiązań typu polyfill, które uzupełniają brakujące w przeglądarce funkcję (wynika z punktu powyżej).
- nie można ładować czcionek w głównej ramce oraz styli CSS, które mogłyby zmieniać wygląd reszty strony.
  Do ładowania styli zalecamy użycie metody `loadCSS()`.
- nie można wywoływać żadnych zewnętrznych usług.
- komponent nie może wprowadzać podatności na ataki do systemu, np. umożliwiać wykonanie kodu przedmioty trzecie czy kradzież sesji.
- komponent powinien działać poprawnie, jeśli zostanie uruchomione wiele jego kopii lub inne komponenty.

  Wyjątek: komponent wielokrotnie zainicjalizowany działa poprawnie, lecz ograniczenia przeglądarek nie pozwala na jego poprawne działanie w większej ilości (np. jeśli widżet korzysta z WebGL)

Interfejs silnika

Bazowy interfejs silnika. Implementacja tego interfejsu jest zawsze wymagana.

```
interface Engine {
    init(container: Element, api, options): Promise<void> | any
    destroy(container)
}
```

- `init(container, api, options)`

  Metoda inicjalizująca komponent

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

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

    - `id` - unikalny identyfikator przypisany do instancji
    - `contrastMode` - jaki tryb kontrastowy wybrał użytkownik
    - `locale` - preferowany język
    - `id` - id uruchomionej instancji. Unikalny.
    - `showAnswers` - czy widżet powinien eksponować prawidłowe odpowiedzi użytkownikowi (poprzez przyciski “pokaż odpowiedź”, “sprawdź”).
    - `files` - tablica plików utworzona przez edytor
    - `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.

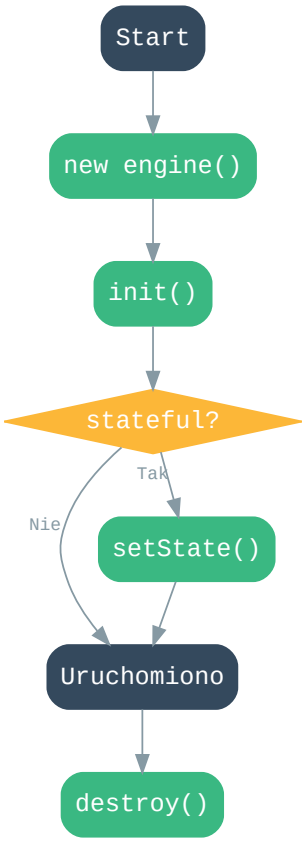
  Obiekt zwracając obietnice, sygnalizuje, że nie zakończył jeszcze inicjalizacji. System wyświetli informację o ładowaniu do czasu ukończenia obietnicy. W przypadku wystąpienia błędu wyświetlony zostanie systemowy komunikat o braku możliwości uruchomienia komponentu.

  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.



Uruchomienie aplikacji.

Stateful

Po zmianie stanu komponent powinien wywołać metodę API `triggerStateSave()`. System wkrótce wywoła metodę `getState()`. Zwrócona wartość zostanie zapisana.

```
interface Stateful {
    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.

```
interface AutoValidable extends Stateful {
    isStateValid(stateData): Boolean
    showStateValidation(isValidationVisible)
}
```

- `isStateValid(stateData)` - Zwraca informację czy stan komponentu `stateData` odpowiada poprawnemu wykonaniu ćwiczenia.
- `showStateValidation(isValidationVisible)` - Wymusza prezentację walidacji stanu (czy ćwiczenie zostało wykonane poprawnie). Metoda wywoływana jest zawsze po metodzie setState.

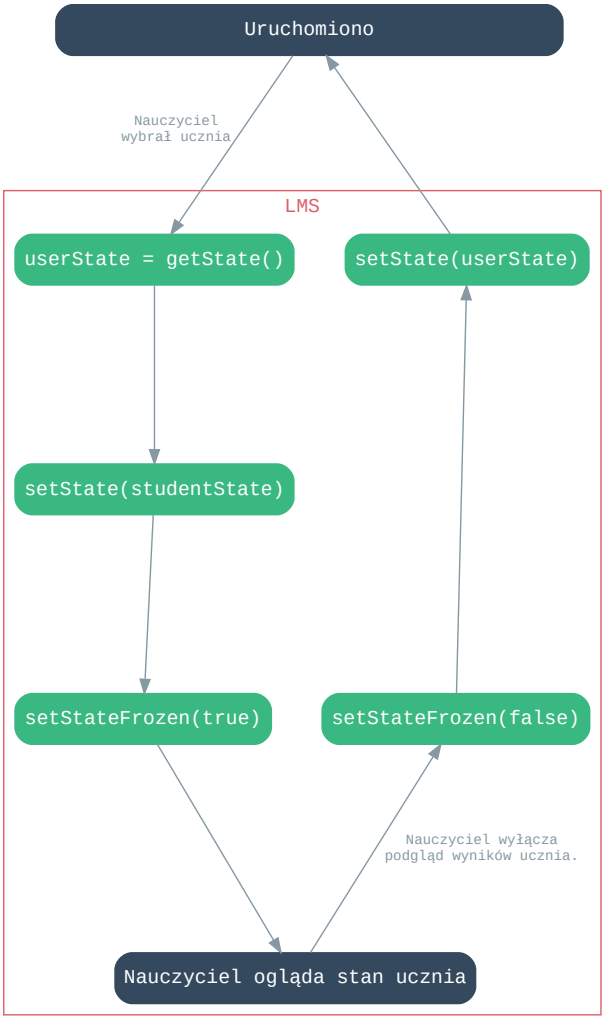
  Po wywołaniu `showStateValidation(true)`, komponent musi zaprezentować walidację stanu. Nie może wtedy zmieniać stanu.

  Po wywołaniu `showStateValidation(false)`, komponent musi ukryć walidację stanu.

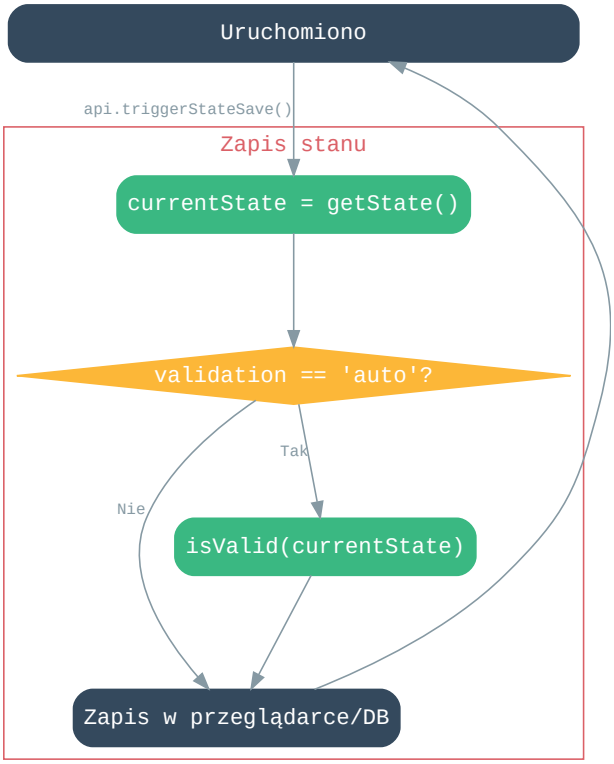
Upload plików przez ucznia

```
interface AllowUserFileUpload extends Stateful {
    getFiles(stateData): Array<string>
}
```

- `getFiles(stateData)` - zwraca tablicę kodów, które mogą być użyte dla obecnego stanu. Pliki wgrane przez ucznia o kodach które nie znajdują się w tej tablicy, zostaną skasowane. Metoda wywoływana jest podczas wywołania `api.uploadFile(fileId, file)`. Jeśli `fileId` nie będzie znajdować się w tej tablicy wgranie pliku przez ucznia nie będzie możliwe.



Współpraca modułu z modułem LMS. Prezentacja odpowiedzi ucznia.



Współpraca modułu z modułem LMS. Weryfikacja poprawności wykonanego zadania.

API

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

```
interface ExerciseApi {

    triggerStateSave(): Promise<void>;
    triggerStateRestore(): Promise<void>;
    enginePath(path): string;
    dataPath(path): string;
    loadCss(realPath): Promise<void>;
    typesetMath(dom): Promise<void>;

    embedWidget(container: Element, manifest: string|object, widgetOptions: object): Promise<void>;
    createWidgetSlot(container: Element, slotName, slotOptions: object, widgetOptions: object): Promise<void>;
    isWidgetSlotFilled(slotName);

    openGallery(images);

    requestFullscreen(container: Element, onFullscreenExit): Promise<void>;
    toggleFullscreen(container: Element, onFullscreenExit): Promise<void>;
    exitFullscreen(): Promise<void>;

    inputFocusIn(input: Element);
    inputFocusOut(input: Element);

    uploadFile(fileId: string, file: File|Blob): Promise<void>;
    removeUploadedFile(fileId: string): Promise<void>;
    removeUploadedFiles(): Promise<void>;
}
```

- `triggerStateSave()` - metoda powinna zostać wywołana po zmianie stanu komponentu.
- `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.
- `loadCss(realPath)` - ładuje plik CSS
- `typesetMath(dom)` - modyfikuje wskazany element DOM. Konwertuje elementy `<math>` tak, aby były poprawnie wyświetlone na wszystkich przeglądarkach.
- `embedWidget(container, manifest, widgetOptions)` - ładuje inną aplikację w elemencie dom `container`.
  W drugim argumencie można podać cały manifest lub identyfikator komponentu. W przypadku podania identyfikatora musi on być zawarty w `manifest.json` w polu `dependencies`.
- `createWidgetSlot(container, slotName, slotOptions, widgetOptions)` - ładuje inną aplikację w elemencie dom `container`. W drugim argumencie podajemy nazwę slotu (dowolna). Slotami zarządza edytor. Jeśli slot został uzupełniony, załadowana zostanie inna aplikacja. W innym przypadku funkcja zwróci błąd (obietnica zostanie odrzucona).
- `isWidgetSlotFilled(slotName)` - zwraca informację czy slot jest uzupełniony.
- `openGallery(images)` - uruchamia systemową galerię zdjęć. W argumencie należy przekazać jeden lub tablicę elementów `<img>`.
- `requestFullscreen(container, onFullscreenExit)` - uruchom kontener w trybie pełnoekranowym.
- `toggleFullscreen(container, onFullscreenExit)` - uruchom kontener w trybie pełnoekranowym. Jeśli jest już w trybie pełnoekranowym, to go opuść.
- `exitFullscreen()` - opuść tryb pełnoekranowy.
- `inputFocusIn(input)` - informuje system, że do elementu formularza należy wyświetlić klawiaturę ekranową.

- `inputFocusOut(input)` - informuje system, że do elementu formularza należy przestać wyświetlać klawiaturę ekranową. **Uwaga:** nie należy bazować tu na zdarzeniu `focus`. W trakcie obsługi klawiatury ekranowej może ona przejmować fokus.

Parametry CSS

Z poziomu CSS, do dyspozycji są następujące zmienne:

- `--font-sans` - systemowy font bezszeryfowy.
- `--font-serif` - systemowy font szeryfowy.
- `--font-mono` - systemowy font o stałej szerokości znaku.

Umożliwiają one wykorzystanie tej samej czcionki, co reszta elementów systemu.

Przykład użycia:

```
.my-class {
    font-family: var(--font-sans);
}
```

Edytor komponentów

Edytor jest osobną aplikacją, z osobnym punktem wejścia (zdefiniowanym w `engine.json`). Umożliwia stworzenie komponentów, które korzystają z utworzonego wcześniej silnika. Jego implementacja nie jest wymagana do poprawnego działania komponentu.

Interfejs edytora

```
interface EngineEditor {

    init(api, options): void
    destroy(): void
    initTab(tab, container: Element, api): Promise<void> | any
    destroyTab(tab, container): void

    setState(stateData): void
    getState(): Object
}
```

- `init(api, options)`

  Metoda inicjalizująca edytor. Należy w niej zadeklarować zakładki, jakie mają być utworzone.

- `destroy()`

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

- `initTab(tab, container, api)`

  Metoda inicjalizująca tab. Jest wywoływana po wywołaniu `init()` lub po `destroyTab()` (w przypadku zmiany zakładki na inną).

- `destroyTab(tab, container)`

  Metoda czyszcząca zakładkę.

  Jej zadaniem jest usunięcie wszystkich elementów DOM, zdarzeń i uwolnienie wszystkich innych zasobów które są używane przez zakładkę.

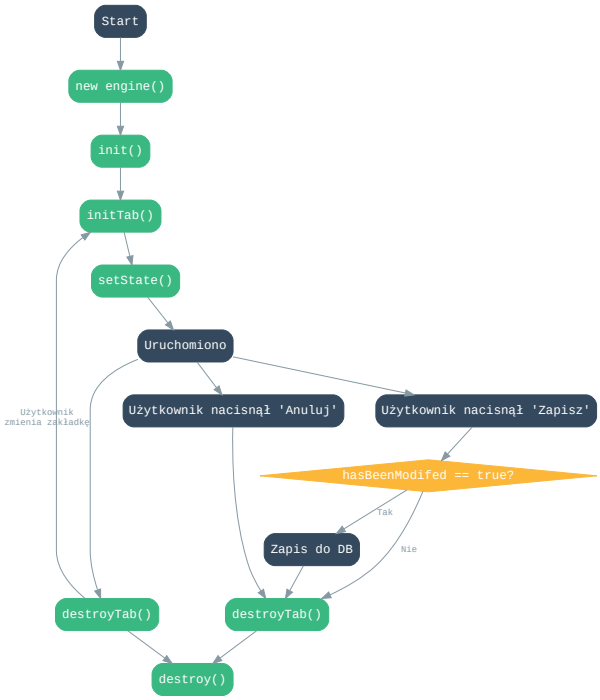
- `setState(stateData)`

  Ustawienie aktualnego stanu dla edytora.

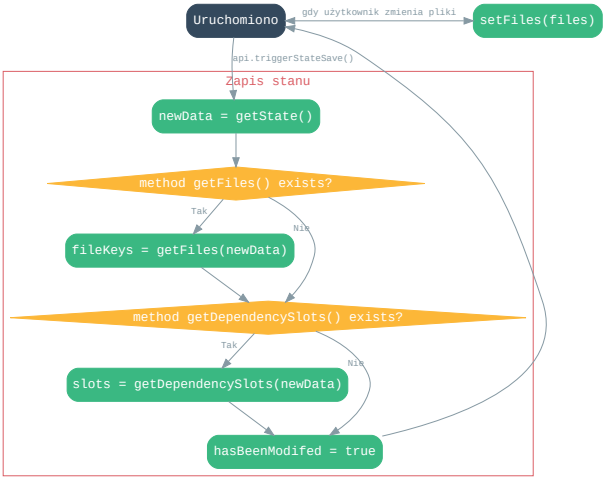
  Przy pierwszym tworzeniu komponentu, do `stateData` przekazana będzie wartość `defaultData` zawartą w engine.json lub NULL.

- `getState()`

  Zwraca aktualny stan edytora. Wartość musi być serializowalna.



Uruchomienie edytora.



Zapis danych w edytorze.

API dla edytora

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

```
interface ExerciseEditorApi {

    addEditorTab(id, name): ExerciseEditorApi;
    triggerStateSave(): Promise<void>;

    enginePath(path): string;
    dataPath(path): string;
    loadCss(realPath): Promise<void>;
    typesetMath(dom): Promise<void>;

    embedWidget(container: Element, manifest, widgetOptions): Promise<void>;
    createWidgetSlot(container: Element, slotName, slotOptions, widgetOptions): Promise<void>;
    isWidgetSlotFilled(slotName): boolean;
}
```

- `addEditorTab(id, name)` - metoda pozwala na utworzenie nowej zakładki.
- `triggerStateSave()` - metoda powinna zostać wywołana po zmianie stanu edytora.
- `createWidgetSlot(container, slotName, slotOptions, widgetOptions)` - ładuje inną aplikację w elemencie dom `container`.
- `isWidgetSlotFilled(slotName)`

Metody `enginePath()`, `dataPath()`, `loadCss()`, `typesetMath()`, `embedWidget()` działają identycznie jak w przypadku API instancji.

Pliki

Interfejs pozwala na zadeklarowanie zapotrzebowania na pliki, które użytkownik tworzący widżet musi wgrać.

```
interface HasFileUploadSupport extends EngineEditor {

    getFiles(state, options) {
        return {
            'KOD': {
                'label': 'Nazwa pliku'
            },
            // ....
        };
    },
    setFiles(files)
}
```

- `getFiles(state, options)` - metoda powinna zwrócić listę plików dla zadanego stanu.
- `setFiles(files)` - metoda wywoływana po wgraniu/usunięciu pliku przez użytkownika. System przekazuje aktualną listę plików.

Pliki dostępne są w instancji w metodzie `init()` w parametrze options.files

```
class ExampleFilesApp {
    init(container, api, files) {
        let url = api.dataPath(files['IMG1']);
        console.debug(
            url ? 'Url pliku:' + url : 'Plik nie został wgrany'
        );
    }
}
```

DependencySlots

```
interface HasDependencySlotsSupport extends EngineEditor {

    getDependencySlots(state)
}
```

Dodatkowy interfejs, wymagany w przypadku użycia metody `createWidgetSlot()`. Metoda `getDependencySlots()` powinna zwrócić wszystkie używane obecnie sloty (tablicę z nazwami), dla danego stanu edytora.

W przypadku zwrócenia niepustej tablicy pojawi się dodatkowa zakładka umożliwiająca uzupełnienie slotu oraz edycję komponentu wewnątrz. Może to być inne ćwiczenie, bądź film.

System w żaden sposób nie sygnalizuje zmiany slotu. Po zmianie jego zawartości przez użytkownika, zawartość kontenera wskazanego w `createWidgetSlot()` zostaje zaktualizowana.

Instancja komponentu

Gotowy silnik można wykorzystać, tworząc instancję komponentu.

Instancję można utworzyć:

- przeciągając komponent z paska narzędziowego, jeśli został dla niego utworzony edytor.

  Uwaga: edytor jest widoczny na pasku narzędziowym, tylko w przestrzeni, w której został utworzony komponent.

- wgrywając paczkę ZIP zawierającą plik manifest.json oraz inne pliki, z których korzysta silnik, a które są specyficzne dla danej instancji.

  Struktura pliku **manifest.json**:

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

  Plik manifest.json musi znajdować się na samym szycie struktury archiwum (nie może znajdować się w żadnym katalogu).

Komponent w wersji drukowanej (PDF)

System, podczas generowania wersji PDF wykorzystuje mechanizmy przeglądarki do wydruku podstrony.

- komponent, który można drukować, należy oznaczyć w engine.json `"printable": true`.
- instancja musi być gotowa do druku, po zakończeniu wywołania `init()` lub po skończeniu zwróconego `Promise`.
- druk musi uwzględniać ustawiony stan ćwiczenia.
- walidacja musi być widoczna w druku.
- można wykorzystywać reguły `@media print` do ukrycia elementów interfejsu.

  ```
  @media print {
      .example {
         display: none;
      }
  }
  ```

Użycie systemowego odtwarzacza multimediów wewnątrz widżetu

- simple *(opcjonalny)* - uproszczony odtwarzacz (tylko odtwórz/zatrzymaj).
- parametry źródła:
  - mediaType **(wymagany)** - rodzaj źródła. Dozwolone wartości: `audio`, `video`.
  - src **(wymagany)** - adres URL źródła w formie mp3 lub mp4.
  - name *(opcjonalny)* - opis wyświetlony tylko w przypadku użycia playlisty.
  - poster *(opcjonalny)* - adres URL pliku `png` lub `jpg`. Obraz zostanie wyświetlony przed uruchomieniem filmu.
  - cam360 *(opcjonalny)* - czy film 360. Obsługiwane jest wyłącznie odwzorowanie walcowe równoodległościowe (equirectangular projection)
- parametry odtwarzacza
  - controls
    - volume: czy pokazywać kontrolę głośności.
    - quality: czy pokazywać wybór jakości (jeśli dostępne).
    - playbackRate: czy pokazywać wybór prędkości odtwarzania.
    - fullscreen: czy pokazywać przycisk do uruchomienia filmu na pełnym ekranie.

```
api.embedWidget(
    container,
    {
        "engine": "core/media-player",
        "data": {
            "simple": false,
            "playlist": [
                {
                    "id": 1,
                    "mediaType": "video" | "audio",
                    "src": source,
                    "name": "Nazwa ścieżki",
                    "cam360": false,
                    "poster": videoPoster,
                }
            ]
        }
    },
    {
        controls: {
            volume: true,
            quality: true,
            playbackRate: true,
            fullscreen: true
        }
    }
)
```

Repozytorium Treści Audiowizualnych

Spis treści

Wymagania techniczne

Materiały audiowizualne wprowadzane do Repozytorium Treści, które stanowi integralną część platformy epodreczniki.pl, powinny mieć jakość źródłową pozwalającą na ich odpowiednią konwersję do różnych formatów emisyjnych akceptowanych i optymalnych z perspektywy urządzeń końcowych użytkowników, które obejmują różne urządzenia stacjonarne i mobilne dostępne na rynku w perspektywie realizacji projektu oraz późniejszych etapów wdrożeniowych.

Dodatkowo, materiały audiowizualne wprowadzane do Repozytorium Treści powinny spełniać następujące wymagania:

Wymagania techniczne dla nowych materiałów graficznych

- dłuższy bok minimum 1600 px
- głębia, poziom kompresji i zakres tonalny zapewniające odbiór ilustracji bez opisanych niżej wad
- przestrzeń kolorów: sRGB
- format pliku:
    - PNG, JPG dla fotografii i ilustracji, przy czym wybór formatu musi uzasadniać stosunek jakości do wielkości pliku (schematyczne ilustracje, bez przejść tonalnych łatwiej skompresować z zachowaniem wysokiej jakości w formacie PNG, a fotografie pełne przejść w formacie JPG)
    - SVG dla grafik wektorowych
- dopuszczalna jest kompresja pliku, która nie wpływa w widoczny sposób na jakość fotografii, dopuszczalne są jedynie niezauważalnie małe artefakty dla powiększenia 1:1 (100%) lub pierwsze oznaki artefaktów małej skali na granicach obiektów w powiększeniu 2:1 (200%),
- całkowicie niedopuszczalne są widoczne straty rozciągłości tonalnej, artefakty na krawędziach obiektów, efekty blokowe, o ile stylistyka fotografii i kontekst dydaktyczny nie uzasadniają wykorzystanie ww. efektów, np. jako element dzieła artystycznego
- obrazy nie mogą być interpolowane, sztucznie powiększane za pomocą programu graficznego
- zdjęcia nie powinny mieć widocznych wad wynikających z nadmiernego wyostrzania, odszumiania, zabrudzeń w torze optycznym, bandingu (pasmowanie, fałszywe kontury), posteryzacji, aberacji chromatycznej.

Wymagania techniczne dla archiwalnych materiałów graficznych

Dopuszczalne jest użycie materiałów archiwalnych, o niższych wymaganiach technicznych, jeśli ich jakość pozwala na realizację zadania dydaktycznego, w związku z którym materiał taki został użyty.
Wymagania i zalecenia funkcjonalne wobec plików graficznych osadzanych w treściach na platformie.

- Obiekty panoramiczne, szczególnie jeżeli mają orientację wertykalną mogą stać się nieczytelne, gdyż algorytmy na platformie skalują je tak, aby zmieściły się na ekranie bez konieczności przewijania.

- Obiekty wektorowe są preferowanym formatem ze względu na skalowalność, nie ma potrzeby konwersji takich obiektów do bitmap.
- Zalecamy unikanie wkomponowywania tekstu w obiekty graficzne ze względu na niespójność czcionki i zalecenia WCAG.

Wymagania techniczne wobec obiektów interaktywnych WOMI osadzanych w treściach na platformie.

- Wzorcowa architektura obiektu interaktywnego:
- Model danych
- Silnik renderujący

Opis komponentów

Model danych wszystkie pliki potrzebne Silnikowi do wyświetlenia Obiektu Interaktywnego w ramach e-podręcznika/e-materiału. Chodzi tu zarówno o dane, parametry jak i obiekty graficzne, dźwiękowe potrzebne do danego zadania interaktywnego.
Silnik renderujący - logika biznesowa, biblioteki JavaScript zdolne przetworzyć Model Danych i wyświetlić go jako Obiekt Interaktywny. Silnik renderujący jest osadzony w bibliotekach platformy i jest przeznaczony do obsługi danej klasy obiektów interaktywnych, np. biblioteki Geogebry, zadań generatorowych itp.

Ogólne wymagania techniczne do materiałów wideo

- kontener MP4
- 1 strumień wideo, kodowany h264
- wielokanałowe strumienie audio kodowane AAC---LC
    - pierwszy strumień audio --- podstawowy (liczba kanałów 2.0 lub 5.1)
    - drugi strumień audio – audiodeskrypcja (liczba kanałów 2.0 lub 5.1)
- napisy
    - nazwane w/g schematu <nazwa_pliku_wideo>_<funkcja>.vtt
    - <funkcja> może oznaczać:
    - subtitles - plik zawiera wyłącznie dialogi
    - captions - plik zawiera dialogi oraz opisy ważniejszych dźwięków i wskazanie kto mówi
    - pliki są w otwartym formacie WebVtt (http://dev.w3.org/html5/webvtt/)
    - nowy materiał wideo powinien uwzględniać możliwość pojawienia się napisów. Należy zarezerwować miejsce na obrazie dla ich prezentacji, zgodnie ze wskazaniem pozycji z plików napisów i nie umieszczać w tym obszarze obrazu innych istotnych informacji.
    - pliki z napisami muszą być precyzyjnie zsynchronizowane czasowo z plikami wideo
    - należy zwrócić uwagę aby pojawiące się napisy nie nakładały się na istniejące już w obrazie wideo podpisy, napisy końcowe i początkowe. Napisy powinny pojawiać się w innym czasie lub być przesunięte w inne miejsce.
    - kodowanie plików UTF8

    - napisy będą prezentowane w aplikacji odtwarzacza opcjonalnie i alternatywnie
- ewentualne efekty 3D w obrazie są wspierane jedynie w poprzez anaglify

Ogólne wymagania techniczne do materiałów audio

Materiały audio powinno być dostarczone w postaci plików MP3.

Szczegółowe wymagania kodowania

Kodowanie h264

- profil High
- liczba klatek/sekundę: 24 (progresywne) lub 25 (progresywnych) lub 30 (dla screencastów)
- rozdzielczość 1920x1080
- przepływność: 3000 kb/s

Kodowanie AAC---LC wielokanałowego strumienia audio

- liczba kanałów: 1.0, 2.0 lub 5.1
- próbkowanie: 48kHz/16bit
- normalizacja poziomu audio: -3dB
- przepływność uzasadniona zawartością nagrania: od 128 (lektor) do 320 kb/s (np. słuchowisko z muzyką) (1.0, 2.0), 640 kb/s (5.1)

Kodowanie MP3 wielokanałowego strumienia audio

- liczba kanałów: 1.0, 2.0
- próbkowanie: 48kHz/16bit
- normalizacja poziomu audio: -3dB
- przepływność uzasadniona zawartością nagrania: od 128 (lektor) do 320 kb/s (np. słuchowisko z muzyką

Kodowanie PCM wielokanałowego strumienia audio

- liczba kanałów: 1.0, 2.0
- próbkowanie: 48kHz/16bit
- normalizacja poziomu audio: - 3dB

Kodowanie materiałów archiwalnych - video

W przypadku samodzielnych materiałów archiwalnych dopuszczalne jest dostarczenie materiałów do Repozytorium Treści w niższej jakości w zakresie rozdzielczości i przepływności. Jeśli materiał źródłowy audiowizualny jest dostępny w formie z intelinią, to należy go odpowiednio przetransformować do postaci progresywnej. Ponadto, dla materiałów archiwalnych należy znormalizować audio. Materiały archiwalne powinny być odpowiednio opisane metadanymi wskazującymi na wszystkie rozbieżności w odniesieniu do wzorcowych wymagań dla materiałów audiowizualnych.

Ogólne wymagania techniczne do obiektów interaktywnych

Obiekty interaktywne (animacje, aplikacje, gry) powinny być stworzone w HTML5 przy wykorzystaniu JavaScript i CSS. Obiekty interaktywne muszą zapewniać responsywność i kompatybilność z:

- Firefox (ostatnia wersja ---1)
- Chrome (ostatnia wersja ---1)
- Opera (ostatnia wersja ---1)
- Microsoft Edge 16
- Safari 12 +
- iOS Safari 12 +

Ogólne zalecania realizacyjne dla produkcji materiałów

Standardy produkcji i redakcji dla materiałów audio

dobór lektorów

- do nagrania zaangażowany powinien zostać profesjonalny aktor lub lektor zatrudniony przez studio nagraniowe lub bank głosów;
- profesjonalnego lektora cechować powinny: poprawna dykcja oraz artykulacja, brak wad wymowy, doświadczenie w naczytywaniu voiceoverów do produkcji telewizyjnych lub filmowych;

wytyczne dla rejestracji i realizacji dźwięku w nagraniach lektorskich (odnośnie lektur, ćwiczeń):

- nagrania lektorów powinny zostać zrealizowane w profesjonalnym studiu nagraniowym;
- elementy udźwiękowienia alternatywnie można kupić w postaci multimedialnej biblioteki dźwięków;
- poziom muzyki wykorzystywanej jako tło w materiale powinien umożliwiać swobodne rozumienie wypowiedzi aktorów lub tekstu lektorskiego;
- dostawca musi posiadać prawa autorskie lub stosowną licencję do wszelkich materiałów zewnętrznych wykorzystywanych w swoich produkcjach;
- we wszelkich nagraniach dźwiękowych nie mogą być słyszalne odgłosy nie będące przedmiotem nagrania (odgłosy tła);

Dodatkowe wytyczne dot. materiałów video

- niedopuszczalne jest ukazywanie w zrealizowanych materiałach filmowych logotypów lub inna forma lokowania produktów komercyjnych (nie dotyczy materiałów archiwalnych);
- dostawca musi posiadać prawa autorskie lub stosowna licencję do wszelkich materiałów zewnętrznych wykorzystywanych w swoich produkcjach;

Wytyczne dla montażu

- dynamika montażu w materiale powinna być dostosowana do charakteru materiału filmowego;
- główną wytyczną podczas procesu montażu jest jak najdokładniejsze i zarazem najciekawsze ukazanie tematu filmu (np. eksperymentu);
- w celu jak najlepszego przedstawienia tematu filmu, zaleca się wykorzystywanie zdjęć zróżnicowanych pod względem kadrów, zrealizowanych w sposób poprawny (statyczne zdjęcia nie mają prawa drżeć, zdjęcia ruchome muszą być wykonywane w sposób harmonijny);

Oprawa graficzna (w tym animacja)

- zalecane jest, żeby w każdym materiale wykorzystywane były jednolite elementy infograficzne (napisy, plansze z infografiką) pod względem wielkości, czcionki itd., elementy graficzne mogą różnić się kolorystyką na różnych poziomach kształcenia;
- umiejscowienie poszczególnych elementów graficznych, szczególnie tekstu na ekranie powinno uwzględniać tzw. obszar bezpieczny;
- konieczne jest opracowanie typograficzne i graficzne plansz, określenie czcionki i dostarczenie jej z polskimi znakami diakrytycznymi (dla materiałów w tym języku);

Zaawansowane alternatywy statyczne WOMI

Spis treści

Wprowadzenie

Dokument opisuje sposób reprezentacji zaawansowanych WOMI w formatach statycznych (PDF, ODT, aplikacje mobilne), statycznych monochromatycznych (EPUB) oraz w formie druku wypukłego (np. alfabet Braille'a).

Struktura plików w ZIP

Alternatywy statyczne zaawansowanego WOMI są zapisane w jednym pliku ZIP, który zawiera trzy katalogi:

ALTERNATIVE-STATIC - opisujący alternatywę dla formatów statycznych

ALTERNATIVE-STATIC-MONO - opisujący alternatywę dla formatów statycznych monochromatycznych

ALTERNATIVE-RELIEF-PRINTING - opisujący alternatywę dla druku wypukłego (np. alfabet Braille'a)

Struktura wszystkich katalogów jest taka sama. W każdym z nich znajdują się (lub mogą się znajdować) następujące pliki:

Plik(i)	Obowiązkowy?	Opis
metadata.xml	T	patrz dalej
content.html (dla STATIC i STATIC-MONO) albo content.xml (dla RELIEF-PRINTING)	T	Zawartość osadzana w miejscu wystąpienia WOMI
solution.html (dla STATIC i STATIC-MONO) albo solution.xml (dla RELIEF-PRINTING)		Zawartość osadzana w sekcji odpowiedzi do zadań
***.png**		
***.jpg**		
***.tiff**		
***.svg**		za wyjątkiem katalogu ALTERNATIVE-STATIC-MONO

W ZIP nie ma potrzeby umieszczania obrazków WOMI zagnieżdżonych. Umieszcza się wyłącznie obrazki niebędące WOMI.

W katalogu ALTERNATIVE-STATIC-MONO pliki graficzne mogą występować wyłącznie w odcieniach szarości oraz w maksymalnej rozdzielczości 530 x 750 pikseli.

Pliki metadata.xml

W każdym w/w katalogu z alternatywą znajduje się dokładnie jeden plik metadata.xml, który opisuje wykorzystane obrazki (umieszczone w ZIP) oraz inne WOMI.

Elementy /alternative-metadata/womis/womi stanowią wylistowanie WOMI wykorzystanych w tej alternatywie; id WOMI są umieszczone w atrybucie womi/@id.

Elementy /alternative-metadata/images/image stanowią wylistowanie obrazków niebędących WOMI wykorzystanych w tej alternatywie; numeryczne id obrazków są umieszczone w atrybucie image/@id,

natomiast atrybut image/@filename definiuję nazwę pliku znajdującego się w tym samym katalogu, co metadata.xml.

Przykład:

```
<?xml version="1.0" encoding="UTF-8"?>
<alternative-metadata version="1.0">
  <womis>
    <womi id="12345" />
    <womi id="4533" />
  </womis>
  <images>
    <image id="123" filename="abc.png" author="Johnny Bravo" licence="CC BY 1.0" />
    <image id="456" filename="uryetwe.jpg" author="Barack Obama" licence="GNU FDL"
/>
    <image id="789" filename="treyui.png" author="Chuck Norris" licence="CC BY 2.0"
/>
  </images>
</alternative-metadata>
```

Inny przykład - gdy nie da się zaprezentować zadania w tej wersji, np. na czarno-białym czytniku (w takim przypadku wyjątkowo w katalogu z daną alternatywą występuje tylko metadata.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<alternative-metadata version="1.0">
  <no-alternative-reason>Na Twoim urządzeniu nie da się rozróżnić kolorów na
obrazku.</no-alternative-reason>
</alternative-metadata>
```

Schemat do walidacji w/w XMLi:

Expand source

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="alternative-metadata">
        <xs:complexType>
            <xs:choice>
                <xs:element name="no-alternative-reason" type="xs:string"/>
                <xs:sequence>
                    <xs:element name="womis" minOccurs="0">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="womi" maxOccurs="unbounded"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="images" minOccurs="0">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="image" maxOccurs="unbounded"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:choice>
        </xs:complexType>
        <xs:unique name="womi-id">
            <xs:selector xpath="womis/womi"/>
```

```
            <xs:field xpath="@id"/>
        </xs:unique>
        <xs:unique name="image-id">
            <xs:selector xpath="images/image"/>
            <xs:field xpath="@id"/>
        </xs:unique>
        <xs:unique name="image-filename">
            <xs:selector xpath="images/image"/>
            <xs:field xpath="@filename"/>
        </xs:unique>
    </xs:element>
    <xs:element name="womi">
        <xs:complexType>
            <xs:attribute name="id" type="xs:integer" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="image">
        <xs:complexType>
            <xs:attribute name="id" type="xs:integer" use="required"/>
            <xs:attribute name="filename" type="xs:string" use="required"/>
            <xs:attribute name="author" type="xs:string" use="required"/>
            <xs:attribute name="licence" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Wymogi dla plików *.html i *.xml

Wymogi ogólne dla *.html

Pliki content.html oraz solution.html należy tworzyć według następujących zasad:

- format XHTML z div'em jako root-elementem
- nie używać atrybutów style i class (!!) - stylowanie elementów odbywa się za pomocą atrybutu ep:role (patrz dalej); od zakazu użycia style istnieje jeden wyjątek (patrz dalej)
- nie używać URLi i ścieżek do plików - osadzanie grafiki i WOMI - patrz dalej
- tytuł zadania powinien zostać umieszczony w content.html tylko i wyłącznie w atrybucie data-title głównego div'a
- nie wstawiać linków do innych elementów podręcznika, w szczególności do odpowiedzi do danego zadania - linki do/z odpowiedzi zostaną wygenerowane przez platformę e-podręczniki.pl
- zawartość kodowana w UTF-8
- nie należy stosować nazwanych encji html -np. , należy stosować encje html dziesiętne -
- treść alternatywna WOMI typu zadanie interaktywne (osadzonego w zadaniu) powinna być zawarta w plikach content.html i solution.html, zgodnie z opisem w sekcji **Przykładowe elementy XHTML**
- treść alternatywna WOMI typu niezadaniowego (nie osadzonego w zadaniu, np. dźwięk, wideo, ilustracja) powinna być zawarta tylko w pliku content.html, zgodnie z opisem w sekcji **Przykładowe elementy XHTML**

Atrybuty ep:role (w XHTML)

Atrybut ep:role jest separatorem semantycznego opisu danego elementy w strukturze xhtml od prezentacji elementu przy pomocy klasy css. Wartość ep:role jest mapowana na klasę css w docelowym formacie emisyjnym (HTML, PDF, EPUB). Dzięki temu zmiana w klasach css nie powoduje zmiany w logice podręcznika - zmienia się tylko mapowanie css.

Przykłady poniżej.

Osadzanie grafiki i innych WOMI w *.html

W kodzie HTML w tagach <img> nie używa się atrybutu src, a odpowiedniego atrybutu data-* (patrz dalej).

W przypadku osadzania innego WOMI skrypty transformujące do formatów statycznych dodadzą **etykietę** i **tytuł** obrazka w div pod img oraz atrybut **alt** do img na podstawie danych z Repozytorium Treści.

Istnieje możliwość ukrycia tytułu przy pomocy atrybutu **data-hide-caption**. Można zdefiniować szerokość obrazka przy pomocy **data-width** (domyślnie w stylu CSS dla alternatywy MONO stosowane jest width:100%). Nie ma możliwośći nadpisania tekstu alternatywnego inline w XHTML.

Inne WOMI osadza się w następujący sposób :

```
<img data-womi-id="12345" data-hide-caption="all|none|label" data-width="50%"/>
```

gdzie "12345" jest identyfikatorem innego WOMI w Repozytorium Treści. W alternatywach wykorzystywać można wyłącznie WOMI, które mają obrazkowe formaty emisyjne.

Obrazki niebędące WOMI muszą być osadzone w container, **tytuł** (jeśli potrzebny) musi być wprost osadzony w markup. Jeśli platforma ma wygenerować **etykietę** dla tytułu (np. Obiekt multimedialny.), należy zawrzeć **div ep:role="label"**. Można zdefiniować szerokość obrazka przy pomocy **style="max-width**:...." (to jedyne miejsce, gdzie można użyć style), przy braku wprost podanego stylu, domyślnie dla alternatywy MONO stosowane jest width:100%.

Dla tego typu obrazków platforma podmieni tylko **data-image-id** na **src** ze ścieżką do pliku i div ep:role="label" na etykietę.

(atrybut alt w tym img jest obowiązkowy, div ep:role="nowomi-image-container" obowiązkowy, div ep:role="nowomi-image-caption" opcjonalny, div ep:role="nowomi-image-label" opcjonalny, div ep:role="nowomi-image-title" opcjonalny, style="width:..." opcjonalny)

```
<div ep:role="nowomi-image-container">
        <img data-image-id="45678" alt="tekst alternatywy obrazka"
style="width:75%;"/>
        <div ep:role="nowomi-image-caption">
                <span ep:role="nowomi-image-label"/>
                <span ep:role="nowomi-image-title">Tytuł</span>
        </div>
</div>
```

gdzie "45678" jest unikalnym (w kontekście rozpatrywanego WOMI silnikowego) identyfikatorem obrazka nadanym przez autora (WOMI silnikowego). W pliku metadata.xml znajduje się mapowanie tego identyfikatora na nazwę pliku graficznego w ZIP.

Uwaga! Zmiany, które zostaną dokonane w dostarczonym kodzie XHTML dla obrazków przed jego wklejeniem do formatu statycznego:

- zamiana atrybutów data-*-id na atrybuty src,
- zamiana atrybutów ep:role na class
- dodanie atrybutu alt dla osadzonych WOMI (nie dla obrazków niebędących WOMI).

Wymogi ogólne dla *.xml

Pliki content.xml oraz solution.xml należy tworzyć według następujących zasad:

- zawartość kodowana w UTF-8
- (...)

Przykładowe elementy XHTML

Do wykorzystania w plikach z kodem xhtml do osadzenia w formatach statycznych (content.html, solution.html)

Szkielet treści zadania

```
<div xmlns:ep="http://epodreczniki.pl/" ep:role="alternative-root" data-
title="Tytuł zadania">
        <div ep:role="problem">Treść polecenia... (możliwe użycie poniższych
elementów XHTML)</div>
        <div ep:role="commentary commentary-example">Treść przykładowego
rozwiązania (możliwe użycie poniższych elementów XHTML, element opcjonalny)</div>
</div>
```

Szkielet rozwiązania

```
<div xmlns:ep="http://epodreczniki.pl/" ep:role="alternative-root">
        <div ep:role="solution">Treść odpowiedzi (możliwe użycie poniższych
elementów XHTML)</div>
        <div ep:role="solution">(element opcjonalny, gdy zadanie ma alternatywną
odpowiedź)</div>
        <div ep:role="commentary">Treść rozwiązania (możliwe użycie poniższych
elementów XHTML, element opcjonalny)</div>
</div>
```

Szkielet treści alternatywy WOMI innego typu niż zadanie (np. film, dźwięk, ilustracja)

```
<div xmlns:ep="http://epodreczniki.pl/" ep:role="alternative-root" data-
title="Tytuł zadania">
        <!-- wszystkie elementy XHTML dostępne poniżej -->
</div>
```

Paragraf tekstu

```
<div ep:role="para">Tutaj proszę podać opis problemu.</div>
```

Elementy tekstowe

```
<em ep:role="italics">ważne</em>
<em ep:role="bold">ważne</em>
<em ep:role="bold italics">bardzo ważne</em>
<dfn ep:role="term">Office Open XML Document</dfn>
<sup ep:role="sup">3</sup>
<sub ep:role="sub">3</sub>
<span ep:role="newline"><br/></span>
<span ep:role="quote">klient w krawacie jest mniej awanturujący się</span>
<span ep:role="writing">Pan Tadeusz</span>
<span ep:role="person">Adam Mickiewicz</span>
<span ep:role="event-name">Bitwa pod Grunwaldem</span>
<code ep:role="code"><span ep:role="label">[PYTHON]</span> print 'Hello
world'</code>
```

Lista numerowana

```
<div ep:role="list">
        <ol ep:role="enumerated">
                <li ep:role="item">
                        <span ep:role="item-decoration">1)</span>
                        Element <em ep:role="italics">italic</em>
                </li>
                <li ep:role="item">
                        <span ep:role="item-decoration">2)</span>
                        <div ep:role="nowomi-image-container">
                                <img data-image-id="45678" alt="Alternatywny tekst
obrazka" />
                                <div ep:role="nowomi-image-caption">Tytuł</div>
                        </div>
                </li>
        </ol>
</div>
```

Lista nienumerowana

```
<div ep:role="list">
    <ul ep:role="bullet">
        <li ep:role="item">
            Element <em ep:role="bold">bold</em>
        </li>
        <li ep:role="item">
            <img data-womi-id="12345" />
        </li>
    </ul>
</div>
```

Tabela

```
<div ep:role="table">
    <table summary="W poszczególnych wierszach tabeli opisano pierwiastki
chemiczne">
        <caption ep:role="table-text">
            <span ep:role="label">Tabela.</span> <span ep:role="title">Gęstość
wybranych pierwiastków chemicznych</span>
        </caption>
        <thead>
            <tr>
                <th scope="col">Nazwa</th>
                <th scope="col">Symbol</th>
                <th scope="col">Gęstość (g/cm<sup ep:role="sup">3</sup>)</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>Wodór</td>
```

```
                <td>H</td>
                <td>0,00008988</td>
            </tr>
            <tr>
                <td>Hel</td>
                <td>He</td>
                <td>0,0001785</td>
            </tr>
        </tbody>
    </table>
</div>
```

MathML

w tekście:

```
<mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML">
    <mml:mfrac>
        <mml:mrow>
            <mml:mn>1</mml:mn>
        </mml:mrow>
        <mml:mrow>
            <mml:mn>2</mml:mn>
        </mml:mrow>
    </mml:mfrac>
</mml:math>
```

jako osobny blok tekstu:

```
<div ep:role="equation">
    <div ep:role="equation-contents">
        <mml:math xmlns:mml="http://www.w3.org/1998/Math/MathML"> ... </mml:math>
    </div>
</div>
```

Przykładowe elementy XML

Szkielet content.xml (element alternative-root dodano, żeby xml był well-formed, docelowo będzie wycinany)

```
<alternative-root xmlns:ep="http://epodreczniki.pl/">
        <title>Tytuł zadania (plain text)</title>
        <problem>Treść polecenia... (możliwe użycie poniższych elementów
XML)</problem>
        <commentary type="example">Treść przykładowego rozwiązania (możliwe użycie
poniższych elementów XML, element opcjonalny)      </commentary>
</alternative-root>
```

Szkielet solution.xml

```
<alternative-root xmlns:ep="http://epodreczniki.pl/">
        <solution>Treść odpowiedzi (możliwe użycie poniższych elementów
XML)</solution>
        <solution>(element opcjonalny, gdy zadanie ma alternatywną
odpowiedź)</solution>
        <commentary>Treść rozwiązania - jeśli jest potrzeba dokładniejszego
opisania odpowiedzi (możliwe użycie poniższych elementów XML, element
opcjonalny)</commentary>
</alternative-root>
```

Szkielet treści alternatywy WOMI innego typu niż zadanie (np. film, dźwięk, ilustracja)

```
<alternative-root xmlns:ep="http://epodreczniki.pl/">
        <!-- wszystkie elementy XML dostępne poniżej -->
</alternative-root>
```

Paragraf tekstu

```
<para>Tutaj proszę podać opis problemu (możliwe użycie poniższych elementów
XML).</para>
```

Osadzanie grafiki i WOMI

Osadzanie innych WOMI (ep:hide-caption opcjonalne):

```
<ep:reference ep:id="123">
        <ep:hide-caption>all|none|label</ep:hide-caption>
</ep:reference>
```

Osadzanie grafiki niebędącej WOMI (longdesc opcjonalne, ep:label opcjonalne - jeśli obecne, wygeneruje się etykieta np. Obiekt multimedialny):

```
<media id="234" alt="tekst alternatywny">
        <longdesc ep:label="true">Tytuł</longdesc>
</media>
```

Elementy tekstowe

(wewnątrz tych elementów tylko plain text)

```
<emphasis effect="italics">ważne</emphasis>
<emphasis effect="bold">ważne</emphasis>
<emphasis effect="bolditalics">bardzo ważne</emphasis>
<term>Office Open XML Document</term>
<sup>3</sup>
<sub>3</sub>
<newline/>
<quote display="inline">klient w krawacie jest mniej awanturujący się</quote>
<ep:writing>Pan Tadeusz</ep:writing>
<ep:person>Adam Mickiewicz</ep:person>
<ep:event-name>Bitwa pod Grunwaldem</ep:event-name>
<code lang="PYTHON">print 'Hello world'</code>
```

Lista numerowana

```
<list list-type="enumerated" number-style="arabic|lower-alpha" mark-suffix="."
start-value="3">
                <item>
                        Element <emphasis effect="italics">italic</emphasis>
                </item>
                <item>
                        <ep:reference ep:id="123"/>
                </item>
</list>
```

Lista nienumerowana

```
<list list-type="bulleted">
        <item>
            Element <emphasis effect="bold">bold</emphasis>
        </item>
        <item>
```

```
            <!-- tu będzie element opisujący image (niebędący WOMI) -->
        </item>
</list>
```

Tabela

```
<table summary="W poszczególnych wierszach tabeli opisano pierwiastki chemiczne">
    <title>Gęstość wybranych pierwiastków chemicznych</title>
    <tgroup cols="3">
        <colspec colnum="1" colname="c1" />
        <colspec colnum="2" colname="c2" />
        <colspec colnum="3" colname="c3" />
        <thead>
            <row>
                <entry>Nazwa</entry>
                <entry>Symbol</entry>
                <entry>Gęstość (g/cm<sup>3</sup>)</entry>
            </row>
        </thead>
        <tbody>
            <row>
                <entry>Wodór</entry>
                <entry>H</entry>
                <entry>0,00008988</entry>
            </row>
            <row>
                <entry>Hel</entry>
                <entry>He</entry>
                <entry>0,0001785</entry>
            </row>
        </tbody>
    </tgroup>
</table>
```

MathML

Jak dla XHTML powyżej.