ZPE Developer Toolkit - ZPE Port

Zadaniem pakietu zpe-port jest dostarczenie interfejsu komunikacyjnego między aplikacją a platformą ZPE. Pakiet udostępnia funkcje do inicjalizacji aplikacji, uruchamiania jej z danymi stanu, zapisywania stanu oraz niszczenia aplikacji.

Pakiet zpe-port jest częścią ZPE Developer Toolkit, który zawiera również pakiet zpe-emulator do testowania aplikacji lokalnie oraz pakiet zpe-editor będący edytorem danych dla nauczycieli.

Zalecenia dotyczące tworzenia aplikacji

- Oddzielamy logikę aplikacji od treści. Wszystkie treści, dane, linki itp. mają znajdować się w pliku scenario.json. Jeżeli w przyszłości trzeba będzie poprawić literówkę w zadaniu, dodać zadanie, dodać nową treść lub skasować istniejącą to edycji będzie podlegać tylko plik scenario.js, schema.json, engine.json i pliki w folderze static.
- Utworzenie pliku scenario.json jest obowiązkowe.
- Uzupełnienie pliku engine.json jest obowiązkowe.
- Utworzenie pliku schema.json który opisuje strukturę danych w pliku engine.json sekcja editor/defaultData jest obowiązkowe jeżeli aplikacja ma mieć edytor dla nauczyciela.
- Struktura pliku schema.json jest ściśle określona
- Struktura engine.json jest określana przez autora platformy
- Wszystkie dane który może zmieniać nauczyciel za pomocą edytora maja znajdować się w engine.json (sekcja editor/defaultData i editor/demoData), oraz opisane w pliku schema.json. Dane te są potrzebne do zbudowania edytora.
- Dane które nie mogą być zmieniane przez nauczyciela (np. treść zadań, teksty dialogów itp.) mają znajdować się w pliku scenario.json
- Struktura pliku scenario.json ma być logiczna i samo opisująca się. Ma być łatwo zrozumiała dla osoby, która będzie później edytować ten plik.
- Aplikacja ma działać jako moduł AMD. Plik webpack.config.js jest już przygotowany.
- Edytor dla nauczyciela będzie dostarczony jako gotowy pakiet. Trzeba go tylko dodać do swojej aplikacji. Plik webpack.config.js jest już przygotowany.
- Style CSS kompilujemy jako moduł/moduły w trybie local. Niedozwolone jest stosowanie globalnych styli CSS lub nadpisywanie styli globalnych (np. body, div, h1 itp.)
- Dane w editor/defaultData i editor/demoData muszą być tymi samymi danymi.
- **NIE** mieszamy kodu z treścią. Treść ma być w pliku scenario.json
- **NIE** wkładamy pliku scenario.json do pliku wynikowego, ma zostać jako osobny plik. To samo dotyczy schema.json i engine.json oraz innych

zasobów tj. images, glb, fbx itd.

- W pliku scenario.json nie mogą znajdować się dane binarne (np. obrazy w formacie base64). Wszystkie zasoby binarne mają być w folderze static i do nich ma być odwołanie w pliku scenario.json
- Jeżeli projekt da się zrobić w oparciu HTML+SVG DOM bez bibliotek 2D/3D to takie rozwiązanie jest mocno rekomendowane, uprości to implementację WCAG.
- Jeżeli aplikacja działa na engine 2D/3D to najlepiej UI zrobić na HTML+SVG DOM (DOM elementy nad canvas 2d/3d), uprości to implementację WCAG (np. focus, tabIndex, role itp. działają od razu).
- W przypadku aplikacji z 3D należy kompresować tekstury do KTX2, również te ukryte w plikach GLB. Desktopy wytrzymają dużo, ale smartphone można wykończyć bardzo szybko prostymi grafikami bez kompresji (PNG czy JPG to nie jest kompresja z punktu widzenia GPU)

Inne Pakiety ZPE Developer Toolkit

- Pakiet zpe-emulator – zajmuje się ładowaniem aplikacji analogicznie tak jak to wykonuje platforma, udaje mechanizmy platformy tj. zapis do profilu, odczyt z profilu, odczyty danych z które może modyfikować nauczyciel
- Pakiet zpe-editor – to edytor danych dla nauczyciela. Edytor jest wspólny dla wszystkich aplikacji i do jego poprawnego działania konieczne jest stworzenie pliku schema.json który opisuje strukturę danych z pliku engine.json sekcja editor/defaultData – o plikach będzie dalej.

  W pakietach staramy się nie grzebać. Jeżeli coś trzeba dajcie znać, postaram się to szybko dodać, a może się przyda też innym.

Przykład użycia

Aplikacja działa jako moduł AMD który platforma ZPE ładuje, uruchamia i zatrzymuje na żądanie.

```
import * as ZPE from "@/zpe-port";
import { init, run, unload, destroy } from "./app";

export default ZPE.create(init, run, unload, destroy);
```

ZPE.create przyjmuje na wejściu 4 funkcje: init, run, unload i destroy, które definiują cykl życia aplikacji. Wszystkie ww. funkcje będą wywoływane w odpowiednich momentach cyklu życia aplikacji.

Plik app.ts trzeba napisać samodzielnie, wystawiając wyżej opisane metody.

```
init(container: HTMLElement) => Promise<void>;
run(stateData: Record<string, any> | null, isFrozen:
 ↪  boolean) => Promise<void>;
```

```
unload() => Promise<void>;
destroy() => Promise<void>;
```

init - funkcja inicjalizująca aplikację. Na wejściu przyjmuje kontener HTML, w którym aplikacja będzie działać (umieszczać element DOM). Zwraca Promise, który sygnalizuje zakończenie inicjalizacji. Najlepiej jest tutaj załadować dane z plików zewnętrznych (np. scenario.json). Na tym etapie dostępne są dane z DB (te skopiowane z engine.json/editor/defaultData lub te już zmodyfikowane przez edytor), ale nie ma jeszcze stanu aplikacji (savedata). W najprostszym przypadku można zapamietać kontener i zakończyć Promise czekając na run.

run - funkcja uruchamiająca aplikację. Na wejściu przyjmuje dane stanu aplikacji (lub null, jeśli brak stanu). Tutaj aplikacja powinna zbudować interfejs użytkownika w kontenerze przekazanym w init oraz zainicjalizować stan aplikacji na podstawie przekazanych danych stanu. Dane stanu mogą być puste (null) jeśli aplikacja jest uruchamiana po raz pierwszy.

UWAGA!!!

1. Funkcja run może być wywoływana wielokrotnie w trakcie życia aplikacji (np. przy przywracaniu innego stanu). Tak będzie kiedy nauczyciel będzie przeglądał postępy ucznia i przywracał jego stan w swojej aplikacji.
2. Funkcja run przyjmuje drugi argument isFrozen (boolean), który informuje czy aplikacja powinna być uruchomiona w trybie “zamrożonym” (tylko do odczytu). W tym trybie aplikacja nie powinna pozwalać na zmiany stanu przez użytkownika (np. blokować pola formularzy, przyciski itp.). Ten tryb jest stosowany przy przeglądaniu postępów ucznia przez nauczyciela.
3. Dane stanu trzeba traktować jako niepewne - mogą być uszkodzone, w innej wersji lub niekompletne. Jeżeli tak jest, aplikacja powinna sobie z tym poradzić (np. uzupełnić brakujące dane domyślnymi wartościami) lub zignorować dane i zainicjalizować stan domyślny (jeśli dane są całkowicie nie do użytku).

unload - funkcja przygotowująca aplikację do ponownego uruchomienia lub zniszczenia.

destroy - funkcja niszcząca aplikację. Musi usunąć wszystkie elementy z DOM i powinna zwolnić zasoby.

Cykle życia aplikacji

Nastepujące metody definiują cykl życia aplikacji.

- init(container) – inicjalizacja aplikacji

- run(stateData, isFrozen=false) – uruchomienie aplikacji z danymi stanu (lub null)
- unload() – przygotowanie do ponownego uruchomienia lub zniszczenia
- destroy() – zniszczenie aplikacji

Możliwe są następujące scenariusze cyklu życia aplikacji:

1. Aplikacja uruchamiana na rzecz ucznia - metody init i run zostaną wywołane raz na sesję ucznia. Metody unload i destroy mogą być wywołane raz lub wcale (jeśli uczeń zamknie aplikację bezpośrednio).
2. Aplikacja uruchamiana na rzecz nauczyciela (przeglądanie postępów ucznia) - metody init, run, unload mogą być wywołane wielokrotnie w trakcie sesji nauczyciela (np. przy przeglądaniu różnych uczniów lub różnych stanów tego samego ucznia). Metoda destroy może być wywołana raz lub wcale (jeśli nauczyciel zamknie aplikację bezpośrednio). Kiedy aplikacja jest uruchamiana w trybie przeglądania postępów ucznia to metoda run otrzymuje drugi argument isFrozen ustawiony na true.
3. Aplikacja uruchamiana w trybie edytora (jeśli dotyczy) - metody init, run, unload mogą być wywołane wielokrotnie w trakcie sesji edytora (np. edycja danych zmiennych przez nauczyciela w edytorze aplikacji). Metoda destroy może być wywołana raz lub wcale (jeśli edytor zostanie zamknięty bezpośrednio).

Przydatne wskazówki

Najlepiej w „init” załadować podstawowe zasoby, uruchomić serwisy (jeżeli są), zrobić inicjalizację “canvas” i ustawić jego tryb pracy 2D/3D itd. Teraz czekamy na „run”, sprawdzamy „stateData” i tworzymy lub odtwarzamy co trzeba np. jeżeli uczeń już gdzieś był w grze to można pokazać „main menu” z opcja „kontynuuj” dając mu możliwość grania dalej. Oczywiście można też od razu przekierować ucznia do miejsca, gdzie był, z tym że wtedy naprawdę trzeba się dobrze przyglądnąć danym z „saveData” czy są kompletne i poprawne. Staramy się unikać sytuacji, że uczeń dostaje błąd bo dane są uszkodzone lub niekompletne i nie jest wstanie nic zrobić, nawet zacząć od nowa.

W trybie „frozen” odtwarzamy stan w którym był uczeń. Jeżeli aplikacja działa na zasadzie startu od main menu to nie pokazujemy go i od razu ładujemy odtwarzamy stan z „stateData”.

Jak przyjdzie „unload” zwijamy treść z kontenera i czekamy na nowy „run” lub na „destroy”. Jak będzie „run” to budujemy treść, a jak „destroy” to zwijamy pozostałe DOM elementy (np. canvas) i koniec.

Jeżeli dane z „stateData” są uszkodzone, niekompletne lub w starej wersji, to najlepiej jest zignorować je i pokazać „main menu” z możliwością rozpoczęcia

gry od nowa. Drugą opcją jest próba naprawy danych i odtworzenie stanu, ale to może być trudne i czasochłonne.

Zapis stanu aplikacji

Dane ucznia zapisuje za pomocą funkcji setState podając na wejściu obiekt JS.

```
function setState(stateData: Record<string, any>):
 ↪  Promise<void>
```

Przykład zapisywanych danych

```
{ "rebus001": "krokus", "rebus002": "" }
```

Zapis danych w scenariusz prawie zawsze jest zdefiniowany jako automatyczny. Platforma zawsze nadpisuje ostatni zapamiętany zapis.

Nie należy przesadzać z częstotliwością zapisów. Zapisujemy tylko jak stan się zmienił w stosunku do poprzedniego stanu. Przy szybko zmieniających się danych (np. pozycja gracza) najlepiej zastosować zapis w interwałach połączony z interakcjami np. jeżeli gracz przemieszcza się po planszy to zapis pozycji gracza robimy np. co 30s, ale jeżeli podejdzie i podniesie przedmiot to zapis o podniesionym przedmiocie i pozycję gracza zapisujemy od razu.

Jeżeli stan wysyłany jest interwałowo to stan powinien być wysłany też podczas „unload".

> Gdy aplikacja jest uruchomiana w emulatorze to „setState" nie powoduje trwałego zapisania stanu. Stan jest przechowywany tylko w kodzie „zpe-port", oraz wyświetlany jest na konsoli. Jeżeli ten stan chcesz wczytać to trzeba skopiować dane z konsoli i zapisać je w postaci pliku w folderze „data" (jak to wczytać patrz. sekcja Developer) lub wklej skopiowane dane do pliku „data/savedata.json".
>
> **Zapis stanu obowiązuje tylko dla trybu "singleplayer", nie stosujemy go do trybu "multiplayer".**

Struktura plików projektu

Folder data

Folder "data" służy do przechowywania plików używanych w trybie deweloperskim. Pliki te nadpisują domyślne pliki z folderu "static".

- "engine-data.json" - alternatywny plik "engine.json"
- "savedata.json" - domyślny plik stanu aplikacji. Jeśli plik jest pusty lub zawiera "null" lub "undefined", aplikacja otrzyma "null" jako stan (brak

stanu). Symuluje to sytuację, gdy w prawidziwym systemie nie ma zapisanego stanu ucznia. Pliki z różnymi stanami można przechowywać w tym folderze i w razie potrzeby zmieniać nazwę pliku na “savedata.json” lub wskazywać inny plik za pomocą opcji wiersza poleceń.

Pliki z folderu “static”

- Plik „scenario.json” – wszystkie dane zmienne których nie może zmieniać nauczyciel w edytorze, np. treść zadań, teksty dialogów itp. To które dane podlegają edycji w edytorze jest określone w zamówieniu.

```
{
    "config": {
        "version": "1.0"
    },
    "rebuses": {
        "rebus001": {
            "imagePath": "images/rebus-001.jpg",
            "answer": "krokus",
            "hint": "To wiosenny kwiat."
        },
        "rebus002": {
            "imagePath": "images/rebus-002.jpg",
            "answer": "dostawa",
            "hint": "Coś, co przyjeżdża do sklepu."
        }
    }
}
```

- Plik „engine.json” – tutaj znajdują się dane aplikacji z których korzysta platforma i aplikacja. Mamy tu:
  - „entry”: wskazuje plik startowy aplikacji,
  - „stateful”: czy aplikacja będzie chciała zapisywać stan,
  - „useWebGL”: opis jest jasny, ale na tą chwilę nie mam informacji co tak naprawdę powoduje dane true/false
  - „editor/entry”: wskazuje plik startowy dla edytora (edytor ogarnia online-skills)
  - „editor/defaultData”: tu wpisujemy wszystkie dane które może edytować nauczyciel w edytorze. To co powinien edytować wynika z zamówienia.
  - „editor/demoData”: są to dane demonstracyjne, na tą chwilę przyjmujemy, że to są te same dane co „editor/defaultData”
  - „editor/editorIcon”: ikona widoczna w edytorze, powinien dostarczyć to online-skills

```
{
    "entry": "runtime/entry.js",
```

```
    "stateful": true,
    "useWebGL": true,
    "editor": {
        "entry": "runtime/editor-entry.js",
        "defaultData": {
            "message": "Hello, ZPE!",
            "playTime": 120,
            "rebuses": {
                "numberOfRebusesToSolve": 5,
                "mode": "medium",
                "timeLimitPerRebus": 10,
                "rebuses": [
                    {
                        "id": "rebus001",
                        "enabled": true
                    },
                    {
                        "id": "rebus002",
                        "enabled": false
                    }
                ]
            }
        },
        "demoData": {
            "message": "Hello, ZPE!",
            "playTime": 120,
            "rebuses": {
                "numberOfRebusesToSolve": 5,
                "mode": "medium",
                "timeLimitPerRebus": 10,
                "rebuses": [
                    {
                        "id": "rebus001",
                        "enabled": true
                    },
                    {
                        "id": "rebus002",
                        "enabled": false
                    }
                ]
            }
        },
        "editorIcon": "editor-icon.svg"
    }
}
```

- Plik „schema.json”- opisuje strukturę danych w pliku „engine.json” z

sekcji „editor/defaultData”:

```
{
    "types": {
        "difficulty": {
            "type": "string",
            "enum": {
                "easy": "łatwy",
                "medium": "średni",
                "hard": "trudny"
            }
        }
    },
    "properties": {
        "playTime": {
            "type": "number",
            "label": "Całkowity czas gry (w minutach)"
        },
        "rebuses": {
            "type": "object",
            "properties": {
                "numberOfRebusesToSolve": {
                    "type": "number",
                    "label": "Liczba rebusów do rozwiązania"
                },
                "mode": {
                    "type": "difficulty",
                    "label": "Tryb",
                    "help": "Wybierz tryb gry. 'łatwy' -
                     ↪  podpowiedzi do rebusów, 'średni' -
                     ↪  bez podpowiedzi, 'trudny' - bez
                     ↪  podpowiedzi i z ograniczonym czasem
                     ↪  na rozwiązanie każdego rebusu."
                },
                "timeLimitPerRebus": {
                    "type": "number",
                    "label": "Limit czasu na rozwiązanie
                     ↪  każdego rebusu (w minutach)",
                    "help": "Dotyczy tylko trybu 'trudny'."
                },
                "rebuses": {
                    "type": "array",
                    "order": "manual",
                    "item": {
                        "type": "object",
                        "properties": {
                            "id": {
```

```
                    "type": "string",
                    "private": true
                },
                "enabled": {
                    "type": "boolean",
                    "label": "Włączony"
                },
                "answer": {
                    "type": "ref",
                    "path": "sce-
                     ↪  nario.json#/rebuses/{item.id}/answer"
                     ↪  ,
                    "label": "Odpowiedź"
                }
            }
        }
    }
}
```

Wersje językowe (dot. wybranych aplikacji)

Ta część dotyczy tylko aplikacji, które posiadają UI wielojęzykowe.

> Przykłady pochodzą z aplikacji „starter-locale”
>
> Zalecana biblioteka to fluent [https://projectfluent.org], w przykładzie używana jest wersja „fluent-bundle„ [https://github.com/projectfluent/fluent.js/tree/main/flue bundle] Po uzgodnieniu dopuszczalne jest inne rozwiązanie, „fluent” jest domyślnym rozwiązaniem dla edytora.

W pliku “scenario.json” w sekscji „config” należy dodać 2 wpisy:

- “locale” - tablica obiektów z opisem dostępnych lokalizacji
  - “id” - identyfikator lokalizacji (np. pl-PL, en-US)
  - “name” - nazwa lokalizacji (np. Polski, Angielski)
  - “path” - ścieżka do pliku lokalizacji (np. locales/pl-PL.json)
- “localeType” - typ lokalizacji (np. fluent) - inne typy do uzgodnienia

```
{
    "config": {
        "version": "1.0",
        "localeType": "fluent",
        "locale": [
            {
                "id": "pl-PL",
```

```
                "name": "Polski",
                "path": "locales/pl-PL.json"
            },
            {
                "id": "en-US",
                "name": "Angielski",
                "path": "locales/en-US.json"
            }
        ]
    },
    "rebuses": {
        "rebus001": {
            "imagePath": "images/rebus-001.jpg",
            "answer": "rebus001-answer",
            "hint": "rebus001-hint"
        },
        "rebus002": {
            "imagePath": "images/rebus-002.jpg",
            "answer": "rebus002-answer",
            "hint": "rebus002-hint"
        }
    }
}
```

- Plik lokalizacji pl-PL.local wskazywany w „scenario.json”:

```
# pl-PL
locale = pl-PL

# Rebusy

rebus001-answer = krokus
rebus001-hint = To wiosenny kwiat.
rebus002-answer = dostawa
rebus002-hint = Coś, co przyjeżdża do sklepu.
```

- Plik „schema.json”- opisuje strukturę danych w pliku „engine.json” z sekcji „editor/defaultData”. Ten plik jest taki sam jak poprzednio, ale w polu „answer” dodano „local”: true, co oznacza, że wartość będzie pobierana z pliku lokalizacji.

```
{
    "types": {
        "difficulty": {
            "type": "string",
            "enum": {
                "easy": "łatwy",
                "medium": "średni",
```

```
            "hard": "trudny"
        }
    }
},
"properties": {
    "playTime": {
        "type": "number",
        "label": "Całkowity czas gry (w minutach)"
    },
    "rebuses": {
        "type": "object",
        "properties": {
            "numberOfRebusesToSolve": {
                "type": "number",
                "label": "Liczba rebusów do rozwiązania"
            },
            "mode": {
                "type": "difficulty",
                "label": "Tryb",
                "help": "Wybierz tryb gry. 'łatwy' -
                 ↪  podpowiedzi do rebusów, 'średni' -
                 ↪  bez podpowiedzi, 'trudny' - bez
                 ↪  podpowiedzi i z ograniczonym czasem
                 ↪  na rozwiązanie każdego rebusu."
            },
            "timeLimitPerRebus": {
                "type": "number",
                "label": "Limit czasu na rozwiązanie
                 ↪  każdego rebusu (w minutach)",
                "help": "Dotyczy tylko trybu 'trudny'."
            },
            "rebuses": {
                "type": "array",
                "order": "manual",
                "item": {
                    "type": "object",
                    "properties": {
                        "id": {
                            "type": "string",
                            "private": true
                        },
                        "enabled": {
                            "type": "boolean",
                            "label": "Włączony"
                        },
                        "answer": {
```

```
                                "type": "ref",
                                "local": true,
                                "path": "sce-
                                 ↪  nario.json#/rebuses/{item.id}/answer"
                                 ↪  ,
                                "label": "Odpowiedź"
                            }
                        }
                    }
                }
            }
        }
    }
}
```