Dokumentacja biblioteki do wizualizacji CoNDeT

Emil Sroka, Hubert Miziołek, Miłosz Wrzesień

2021

Spis treści

1 Historia dokumentacji

NOTE: do usunięcia w ostatecznej wersji

1.1 Wersja I.0

- Architektura — Iteracja I
  - Moduł prezentacji – ogólne założenia
  - Moduł prezentacji – komponenty
  - Wylistowanie pozostałych modułów

2 Wstęp

TODO: w nieokreślonej wersja

3 Architektura – Iteracja I

Założenia bazowe W celu uniknięcia kolizji nazw z innymi bibliotekami zostanie zastosowany wzorzec `namespace`. Globalny obiekt `CoNDeT` będzie przechowywał poszczególne moduły biblioteki.

3.1 Moduł wspólny

Moduł wspólny będzie znajdował się pod kluczem `core` w obiekcie biblioteki. Będzie on odpowiedzialny za funkcji oraz typy współdzielone między modułami.

NOTE: dokumentacja modułu tworzona na bieżąco wraz z dodawaniem współdzielonych funkcjonalności

3.2 Moduł główny

Moduł spinający pozostałe moduły.

3.3 Moduł danych

Moduł odpowiedzialny za obsługę danych. Będzie znajdował się pod kluczem `data` w obiekcie biblioteki.

TODO: przechowywanie danych i wywoływanie procesu aktualizacji widoku – w wersji II

TODO: wczytywanie i zapisywanie danych do pliku – w wersji II lub III

TODO: interface do modyfikacji danych z poza modułu (np. z modułu UI w trybie edycji) – w wersji III lub IV

3.4 Moduł prezentacji

Moduł prezentacji będzie znajdował się pod kluczem `ui` w obiekcie biblioteki.

3.4.1 Ogólne założenia

Miejsce na stronie Interfejs użytkownika będzie generowany wewnątrz pierwszego podanego elementu `div` o podanym `id`. Jeżeli `id` nie zostanie podane, domyślnie będzie poszukiwany `div` o `id` równym `condet-canvas`.

Technologie Wizualizacja będzie bazowała na elementach HTML (tabele) oraz SVG (połączenia między tabelami). Operacje na elementach będą odbywały się pośrednio poprzez komponenty opisane poniżej.

Obiekt zarządzający Obiektem zarządzającym interfejsem użytkownika będzie `UserInterface`. Będzie on tworzył, aktualizował i usuwał komponenty.

Rozmieszczenie obiektów Obiekty będą rozmieszczone wewnątrz wybranego elementu `div`. Rozmiar powinien być ustalony przez osobę korzystającą z biblioteki. `UserInterface` nada elementowi style (`overflow: hidden;`) ukrywające zawartość wychodzącą poza `div` oraz będzie zapisywał offset. Komponenty będą miały przekazywany aktualny offset (jako parametr funkcji wchodzących w skład cyklu życia komponentu). Bazując na tym komponent powinien sam ustalić swoją pozycję. Umożliwi to przesuwanie widoku za pomocą mechanizmu drag & drop.

Tryby Za pomocą wzorca stan (state design pattern) komponenty oraz obiekt `UserInterface` będą mieć różne zachowania zależnie od stanu. Zasada działania została opisana przy komponentach. Pozwoli to na implementację dwóch trybów: wyświetlania oraz edycji.

Tryby wyświetlania Tryb wyświetlania pozwoli jedynie na wyświetlanie tabeli i połączeń między nimi.

Tryby edycji TODO: w wersji III lub IV

3.4.2 Komponenty

Bazowy komponent Bazową klasą po której będą dziedziczyć komponenty jest `BaseComponent`. Wyodrębnia ona części wspólne wszystkich komponentów. Zawiera metody odpowiedzialne za cykl życia komponentu:

- `init` – metod wywoływana przy utworzeniu komponentu
- `update` – metoda wywoływana przy zmianie stanu aplikacji (np. zawartość tabeli została zmieniona)
- `destroy` – metoda wywoływana przy usuwaniu komponentu

Wymiary obiektu Kolejną grupą metod są funkcja związane z położeniem i rozmiarem komponentu:

- `containsPoint` – metod sprawdza czy podany punkt zawiera się wewnątrz danego komponentu
- `getDimensions` – metoda zwraca wymiary (szerokość i wysokość) komponentu
- `getPosistion` – metoda zwraca pozycję (x i y) komponentu

Obsługa zdarzeń Każdy komponent będzie również zawierał właściwość `state` (wzorzec stan). Konkretna implementacja stanu będzie odpowiadać stanowi w którym jest moduł prezentacji (np. stan prezentacji, stan edycji). Obiekt `State` będzie implementował metody związane z obsługą zdarzeń: `onKeyUp`, `onKeyDown`, `onMouseUp`, `onMouseDown`, oraz dwie własne metody wywoływane przy utworzeniu oraz zniszczeniu danej istanci: `onInit` oraz `onDestroy`. Metody te będą wywoływane przez bazowy komponent podczas wywołania metody `setState` służącej do zmiany stanu (wzorzec odwrócenie sterowania – inversion of control design pattern). Do ustawienia subskrypcji na zdarzenia będzie służyć metoda `setupEventListiners` obecna w `BaseComponent`.

Zarządzanie dziećmi Każdy komponent będzie mógł być skomponowany z innych komponentów (np. tabela może wykorzystywać komponent wiersza i nagłówka). Komponent bazowy będzie zawierał uniwersalną część logiki związaną z tym problemem. Właściwość `children` będzie tablicą dzieci. Do dodania nowego komponentu będzie służyć metoda `appendChild`, natomiast do usunięcia metoda `removeChild`.

4 Format danych

4.1 Zastosowany Format

Do zapisywania i wymiany danych w projekcie użyty zostanie format JSON (JavaScript Object Notation). Pozwala on zapis liczb, tekstu (string) , list, typów logicznych, obiektów oraz null'a. Odpowiednio zapisane i przygotowane dane w tym formacie uproszczą ich obróbkę przy ich wymianie z aplikacją.

4.2 Struktura

Wymiana danych będzie następowała poprzez wymianę pliku w formacie JSON. Każdy przesyłany plik zawiera w sobie listę danych, w której znajdują się obiekty reprezentujące pojedynczą tablicę decyzyjną. Każda tablica zawiera w sobie:

1. swoją nazwę - *name*
2. klasę określającą miejsce w hierarchii - *class*
3. obiekt zawierający koordynaty x i y określający położenie w oknie przeglądarki - *coordinates*
4. obiekt reprezentujący kolumny zawierający 2 listy z nazwami warunków oraz decyzji - *columns*
5. listę wierszy - *rows* zawierającą:
   (a) id danego wiersza - *row_id*
   (b) listę warunków w postaci tablicy dwuelementowej zawierającej id warunku i zawartość komórki - *conditions*

(c) listę decyzji w postaci tablicy dwuelementowej zawierającej id warunku i zawartość komórki - *decisions*

(d) obiekt połączenie wskazujący na nazwę tablicy oraz id wiersza - *connection*

Oszczędność pamięci Zapisanie naszych tablic w ten sposób oszczędza ilość danych poprzez zlikwidowanie potrzeby wysyłania pustych pól tekstowych. Jest to możliwe dzięki nadaniu identyfikacji naszym warunkom oraz decyzjom.

TODO: w wersji II