Önálló laboratórium 2

# Fejlesztői környezet

A projekt TypeScript és Less nyelvű, ezért a projektet fordítani kell. A fordítást a Gulp build rendszer végzi el. Ez úgy van beállítva, hogy magát a dátumválasztót egy fájlba, illetve minden integrációs fájlt külön-külön fájlba fordítson. Ez a config fájl a gulpfile.js.

A tesztek futtatását a Karma + Jasmine keretrendszerek végzik. A Karma configja a karma.conf.js fájlban található.

A projektet a VS Code IDE-ben fejlesztettem. A buildelés és a tesztelés megkönnyebbítése érdekében saját parancsokat definiáltam a .vscode/tasks.json fájlban. Így egy gombnyomással lehet buildelni vagy lefuttatni a teszteket.

# DateSlider

A DateSlider egy vagy több csúszkából álló dátumválasztó komponens. Egy DateSlider példányosításához szükség van egy HTMLElement-re, amibe bele fogja rakni a csúszkákat, illetve a komponens beállításaira.

Látványterv:

## Belső modell



A komponens belső modellje a DateSliderModel. Ez külön-külön propertyben tárolja az év, hónap, nap, óra, perc, másodperc, időzóna adatait. A hónap és a nap 1-től kezdődik. Emiatt szükséges lehet a parszolás és a formatálás, mivel a komponenst használó fejlesztő nem biztos, hogy ilyen formátumú adatokkal akar dolgozni.

Az intervallumkezeléshez és a validációhoz szükséges lehet két belső modell összehasonlítása. Az összehasonlítás egyszerű a tárolás miatt. Elég csak a legnagyobb időegységtől a legkisebb felé haladni, és páronként összehasonlítani a két modell. Ha egy időegységben megegyeznek, csak akkor kell tovább menni a következő kisebb időegységre, különben már eldőlt, hogy melyik nagyobb és melyik kisebb. Akkor ugyanakkorák, ha minden időegységben megegyezik a két modell.

## Parszolás és formatálás

A parszolás akkor történik, amikor a komponensnek be akarunk állítani egy új értéket. Az új érték beállításakor meghívódik a beállításoknál előre megadott parszoló függvény, ami átalakítja a megadott értéket az új belső modellé. Mivel a parszoláshoz nem biztos, hogy elegendő csak az érték, ezért a beállításoknál megadhatók plusz parszolási beállítások is. A komponens tartalmaz előre megírt parszoló függvényeket:

* stringből egy megadott formátum alapján

Plusz beállítások: { format: string }

* Date objektumból

Plusz beállítások: { type: "local" | "utc" }

* UNIX timestamp alapján

Plusz beállítások: { type: "milliseconds" | "seconds" }

Megadható saját parszoló függvény is, ekkor az alábbi szignatúrájú függvényt kell megvalósítania a fejlesztőnek:

* (input: any, options: any) => DateSliderModel

Ahol az input az új érték, az options a beállításoknál megadott parserOptions property és a visszatérési érték pedig a komponens egy belső modellje.

A formatálás akkor történik, amikor a komponensből valamilyen módon megkapjuk az éppen reprezentált dátumot. A formatálás nagyon hasonlóan működik a parszoláshoz és ugyanolyan típusú előre megírt formatáló függvényeket tartalmaz a komponens.

Saját formatáló függvény írásakor az alábbi szignatúrájú metódust kell megvalósítani:

* (input: DateSliderModel, options: any) => any

Ahol az input az komponens belső modellje, az options a beállításoknál megadott formatterOptions property és a visszatérési érték pedig a formatált érték.

## Validáció

A komponens aktuális értékének validálásához megadható minimális, maximális érték, illetve saját validáló függvény. A minimális és maximális értéket a fejlesztő által használt formátumban kell megadni, tehát le fog futni ezekre az értékekre a parszoló függvény, és annak az eredményét fogja a komponens összehasonlítani az épp aktuális értékkel.

Saját validáló függvény megadásakor az alábbi szignatúrájú függvényt kell implementálni:

* (value: any) => boolean

Ahol a value paraméter az éppen validálandó belső modell formatálva, tehát a fejlesztő által használt formátumban. A visszatérési érték pedig true, ha valid az érték, különben invalid.

A validáció után a komponens példányosításakor megadott HTMLElement-re rákerül a "date-slider-valid" vagy a "date-slider-invalid" class a validáció eredményétől függően.

## Callback

Fontos lehet, hogy a komponenst használó fejlesztő értesüljön arról, hogy megváltozott a komponens értéke. Erre használható a callback.onValueChanged beállítás. Megadható egy függvény, ami megkapja a változtatás kontextusát. A kontextus tartalmazza, hogy éppen valid-e a belső modell, illetve az előző értéket és az új értéket.

Minden callback kontextusa a DateSliderEventContext-ből származik le. Ez az osztály tárolja, hogy az event propagáció meg van-e állítva, és ez alapján az eventkezelő megszakítja vagy folytatja a beregisztált callbackek meghívását.

A callbackek tárolását és az eventek tüzelésekori callback hívásokat a DateSliderEventHandler osztály kezeli.

## Csúszkák

Megadható, hogy milyen csúszkákat tartalmazzon a példányosítandó komponens, ez részletesen taglalva lesz lentebb. Ritka, de szükséges lehet a csúszkák közti kommunikáció, mint például a nap csúszka maximuma az évtől és a hónaptól függően változhat. Ezt a komponens automatikusan kezeli.

## Beállítások

A fent taglalt működéshez szükséges beállításokat az alábbi formátumban adhatók meg.

{

value?: any;

sliders?: SliderOptions[];

parser?: "timestamp" | "string" | "date" | ((input: any, options: any) => DateSliderModel);

parserOptions?: any;

formatter?: "timestamp" | "string" | "date" | ((input: DateSliderModel, options: any) => any);

formatterOptions?: any;

callback?: {

onValueChanged?: (context: Context.ValueChangeContext) => void;

};

validation?: {

custom?: (value: any) => boolean;

min?: any;

max?: any;

};

}

Egy komponens példány egyes beállításait meg lehet változtatni az updateOptions és a replaceOptions függvényekkel. Az updateOptions módosítja az eddigi beállításokat az újakkal, tehát ezzel a függvénnyel lehet részleges beállításmódosításokat is végezni. A replaceOptions pedig teljesen lecseréli az eddigi beállításokat az újakra.

### Alapértelmezett beállítások

A komponens alaértelmezett beállításokat tartalmaz, hogy out of the box is jól működjön. Ezek a beállítások módosíthatók, és a módosítás után már minden újonnan példányosított komponens az új alapértelmezett beállításokkal fog inicializálódni.

Az egyes komponenseknél megadott plusz beállítások felülírják az adott komponensnél az alapértelmezett beállításokat.

# Egy csúszka

A komponens akárhány csúszkát tartalmazhat, de csak előre definiált fajtákból, ezek:

* év
* hónap
* nap
* óra
* perc
* másodperc
* univerzális: a dátumot és időt is egyetlen csúszkán ábrázol
* univerzális dátum: a dátumot egyetlen csúszkán ábrázolja
* univerzális idő: 00:00:00-23:59:59-ig ábrázolja az időt egy csúszkán

## Felépítés



Egy csúszka egy kezdőpont és egy végpont között kifeszített szakaszon mutatja az aktuális értéket egy fogantyú elhelyezésével. A fogantyú egérlenyomásra és érintésre mozgathatóvá válik és a felhasználó így módosíthatja az aktuális értéket a kezdő- és végpont között. Alapvetően a csúszka vízszintes fekvésű, de a minimális mozgatási megkötések miatt bármilyen szögben álló, vagy akár folyamatosan forgó komponens is készíthető.

A csúszka aktuális értékének megjelenítéséhez lehet használni a displayValueFormatter beállítási metódust. Ezzel egyedi megjelenítést lehet adni az értéknek. Ha nincs megadva és az alapértelmezett beállításoknál sincs megadva, akkor fallbackként az érték toString metódusából kapott szöveget fogja megjeleníteni.

### Templatezés

A komponenst használó fejlesztő használhatja a beépített, alapértelmezett csúszkakinézetet, természetesen CSS módosításokat alkalmazhat, vagy teljesen felüldefiniálhatja a komponens UI-ját a template beállítás használatával.

Ahhoz, hogy egy saját templatet tudjon kezelni a komponens meg kell adni a template alapjául szolgáló HTML Element-et a beállításoknál. A templatenek az alábbi kényszereket kell teljesítenie, hogy működőképes legyen a csúszka:

* A template elemen belül kell lennie egy "slider-control-template" osztályú elemnek. Erre az elemre "position: relative" és "overflow: hidden" fog kerülni, ami szükséges a csúszka működésére. Ennek az elemnek tartalmaznia kell az alábbi elemeket:
  + Egy elem, aminek "slider-handle-template" osztálya van. Ez lesz a csúszka fogantyúja.
  + Egy elem, aminek "slider-control-start-template" osztálya van. Ez lesz a csúszka kezdőpontja.
  + Egy elem, aminek "slider-control-end-template" osztálya van. Ez lesz a csúszka végpontja.
* Opcionálisan megadható elemek:
  + "slider-value-container-template" osztályú elem, ebbe fogja beleírni az épp aktuális értéket a csúszka.
  + "slider-marker-template" osztályú elem. Ezeket fogja kirakni érték jelölőként a csúszka mellé adott távolságonként. Ha ez az elem meg van adva, akkor szükséges, hogy legyen egy "marker-value-container" osztályú leszármazott eleme, amibe a jelölt értéket fogja beleírni a csúszka.

## Belső modell

Egy csúszka belső modellje egy intervallum, aminek van kezdő- és végértéke, illetve egy aktuális értéke. Az intervallum kezdő- és végértéke változtatható, illetve csúsztatni is lehet az intervallumot, ami annyit tesz, hogy az értékek közti távolság nem változik, mind nő vagy csökken egy adott értékkel.

A belső modell lebegőpontos számokkal dolgozik, de a csúszka magán kívülre csak egész értékeket ad át. Ezt az átváltást a Math.round metódussal végzi el.

Az egyes csúszkák más-más intervallumokkal dolgoznak:

* év: az évszámok az intervallum egyes értékei, például: 2017, 1930.
* hónap: [1, 12] zárt intervallum.
* nap: az évtől és a hónaptól függően [1, 28], [1, 29], [1, 30], [1, 31] zárt intervallumok egyike.
* óra: [0, 23] zárt intervallum.
* perc: [0, 59] zárt intervallum.
* másodperc: [0, 59] zárt intervallum.
* univerzális: Unix timestamp másodpercben.
* univerzális dátum: Unix timestamp napban, tehát 1970-01-01 óta eltelt napok száma.
* univerzális idő: [0, 86399] zárt intervallum. 86400 másodperc van egy napban.

## Csúszka fogantyú mozgatása és érték kiszámolás

A fogantyú mozgatásakor az érték úgy számolódik ki, hogy a csúszka kezdőpontja és a kurzor között kifeszített szakasz merőleges vetületét veszi a csúszka egyenesére. A merőleges vetület hossza és az egyenes hosszának aránya mondja meg, hogy mennyi lesz az új érték.

### Csúszka szélén slide / expand

Lehetőség van extra funkció beállítására, ha a csúszka fogantyúja az intervallum egyik széléhez ér. Alapértelmezetten semelyik csúszka sem csinál semmit az intervallum szélén.

Az alábbi extra lehetőséek vannak:

* Az intervallum megnő
* Az intervallum elmozog az adott irányba



* Egy megadott hosszig megnő az intervallum, utána csak mozog



## Jelölők



A csúszka mellett megjeleníthetők jelölések is amelyek az adott helyen levő értéket mutatják. A jelölők megjelenítéséhez használatos a beállítások markers.showValueMarker, aminek egy metódust megadva lehet szabályozni, hogy mikor jelenjen meg a jelölő és mikor ne. A paraméterként kapott value, minimum és maximum alapján lehet kiszámolni, hogy legyen-e az adott értéken jelölő vagy sem. A visszatérési érték egy osztálynév, vagy osztálynevek tömbje, ami fel fog kerülni az adott értéket mutató jelölőre. Üres string visszaadásakor nem kerül semmilyen osztály a jelölőre, de az meg fog jelenni. Ahhoz, hogy ne jelenjen meg a jelölő, nullt kell visszaadni a metódusban. Azért nem csak egy boolean értékkel van jelölve, hogy megjelenjen-e egy értéken a jelölő, vagy nem, mert így akár minden egyes jelölő egyedivé tehető.

A jelölők úgy lesznek pozícionálva, hogy egy jelölő középpontja merőlegesen egy vonalban legyen a jelölt érték helyével a csúszkán.

A jelölt hely értéke formázható a markers.displayValueFormatter beállítási metódussal. Ha ez nincs megadva, akkor fallbackként a csúszka displayValueFormatter metódusát fogja használni, ha ez sincs megadva, akkor az érték toString metódusából kapott szöveget fogja megjeleníteni.

A csúszka és a jelölő közti távolságot lehet állítani a markers.perpendicularOffset értékkel, ami pixelben megadott értékkel fogja eltolni a jelölőket a csúszkától.



## Callbackek

A csúszka 4 callbacket publikál:

* onValueChanged
* onSliderHandleGrabbed
* onSliderhandleReleased
* onSliderhandleMoved

A példányosításkor minden callbackre csak egy listenert lehet feliratkoztatni, de a már példányosított komponensen van lehetőség további listenereket is feliratkoztatni.

## Beállítások

{

type: "year" | "month" | "day" | "hour" | "minute" | "second" | "universal" | "universal-date" | "universal-time";

movement?: "none" | "slide" | "expand" | "slide expand";

movementSpeed?: number;

movementStep?: number;

expandLimit?: number;

displayValueFormatter?: (value: number) => string;

markers?: {

showValueMarker?: (value: number, minimum: number, maximum: number) => string | string[];

displayValueFormatter?: (value: number, minimum: number, maximum: number) => string;

perpendicularOffset?: number;

};

template?: HTMLElement;

callback?: {

onValueChanged?: (context: Slider.Context.SliderValueChangeContext) => void;

onSliderHandleGrabbed?: (context: Slider.Context.SliderHandleContext) => void;

onSliderHandleReleased?: (context: Slider.Context.SliderHandleContext) => void;

onSliderHandleMoved?: (context: Slider.Context.SliderHandleContext) => void;

};

}

### Alapértelmezett beállítások

A komponens alaértelmezett beállításokat tartalmaz adott típusú csúszkáákra, hogy out of the box is jól működjön. Ezek a beállítások módosíthatók, és a módosítás után már minden újonnan példányosított komponens az új alapértelmezett beállításokkal fog inicializálódni.

Az egyes csúszkáknál megadott plusz beállítások felülírják az adott komponensnél az alapértelmezett beállításokat.

# Tesztelés

A projekt elején úgy döntöttem, hogy kipróbálom a TDD-t és ez a projektett ezzel készítem el. Ez az elején még nagyszerűnek tűnt, a projekt készítése során arra jutottam, hogy nem lehetett a jelenlegi erőforrások mellett tisztán TDD-vel megcsinálni a projektet. Vannak a projektnek részei, amik könnyen tesztelhetőek, ezeket TDD-vel csináltam meg. Viszont Az UI-hez kapcsolódó részek teszeléséhez, mint például a megjelenés, a felhasználói beavatkozásokhoz (egérmozgatás, touch) sokkal komolyabb teszt keretrendszer lenne szükséges, mint amennyit bele tudtam szánni a projetbe.

## Unit tesztek

Az alábbi részek lettek unit tesztelve:

* Parserek
* Formatterek
* Belső modell a dátum tárolásához és összehasznlításához
* Egy csúszka intervalluma

## Integrációs tesztek

Integrációs tesztek lettek írva a parserek és a formatterek közös működésére, hogy ha egy parser beolvas valamit a belső modellbe, akkor a hozzá tartozó formatter ugyanazt a kezdeti értéket fogja visszaadni, mint ami az eredeti bemenet volt.

# Integráció

A DateSlider integrálható különféle keretrendszerekkel. Az integráció fájlokban találhatóak, mint a fő komponens, így csak azt kell a kliensnek letöltenie, amit használni is fog.

## jQuery

A jQuery selectoron meghívható dateSlider metódus példányosít egy DateSlider-t minden kiválasztott elemre. A metódusnak megadható paraméterként a DateSlider(ek) beállításai is. A DateSlider példányt eltárolja a jQuery data metódusával, aminek a "dateSlider" kulcsot megadva lekérhető a példány. A komponens megszűnésekor a jQuery datából is kitörlődik a mentett referencia.

## AngularJS

Az Angular integrációt egy "dateSlider" nevű modul tartalmazza, ezt kell a dependenciák közé rakni az Angular boostrapkor, hogy használható legyen a komponens.

A modul tartalmaz egy "dateSliderOptions" nevű providert, ahol be lehet állítani az alapértelmezett beállításokat.

A modul fő eleme a "dateSlider" nevű direktíva. Izolált scopejába ezeket a paramétereket lehet megadni:

public scope = {

"dateSlider": "=?instance",

"max": "<?",

"min": "<?",

"ngModel": "=",

"options": "=?",

};

Az direktíva használatához szükséges az ngModel direktíva is, ez kezeli a külső modellt és formok használatakor is jól fog kezelődni a komponens, mint form elem.

A min, max és options változását watcholja a direktíva és frissíti a DateSlider példányt az új értékekkel.

A DateSlider értéke kétféleképpen változhat:

* A felhasználó mozgatja a fogantyúkat és így belülről változik meg a kiválasztott dátum / idő. Ekkor a direktíva elkapja a DateSlider példány "onValueChanged" eseményét és beállítja az ngModelnek az új értéket, illetve azt, hogy az új érték valid vagy invalid.
* Az ngModel is változhat a kiválasztott dátum / idő programkódból. Ezért szükséges az ngModel értékének változásait is watcholni, és az új értékkel frissíteni a DateSlider példányt.

Ez a kettős működés viszont alapból problémákat okozhat. Vegyük példaként az alábbi esetet: A felhasználó módosítja az egyik fogantyút, így megváltozik a komponens értéke. Ennek hatására elsül az "onValueChanged". Ezt lekezeli a direktíva úgy, hogy elmenti az új értéket az ngModelbe. Ezután az ngModelre állított watch el fog sülni, mivel az megváltozott. Itt beállítja a DateSlider példányt az új értékre (ami már alapból be van állítva a DateSliderben, mivel onnan jött ez az érték). Ezután a DateSlider újra elsüti az "onValueChanged"-et. Ezt újra lekezeli az eseménykezelő, de az ngModel watchja már nem fog még egyszer elsülni, mivel a második event értéke volt már benne. Ez azt okozta, hogy a fogantyú mozgatáskor ugrált. Ezt kikerülendő az "onValueChanged" eseménykezelőjébe bekerült egy 0-s timeout, ami alapján engedélyezett vagy tiltott az ngModel frissülése utáni értékbeállítás.