hillel IT school

Lesson 38

План заняття

- Material UI
- Formik

Material UI

Material UI – це бібліотека компонентів React з відкритим кодом, яка реалізує Material Design від Google. Він комплексний і може використовуватися у виробництві з коробки. MUI пропонує повний набір безкоштовних інструментів інтерфейсу користувача, які допоможуть вам швидше завантажувати нові функції, але є і багато платного контенту. Також в неї є вже готові темплейти для веб сайтів.

Веб сайт https://mui.com/

Github https://github.com/mui/material-ui

Готові темплейти https://mui.com/material-ui/getting-started/templates/

Material UI installation

Щоб використовувати бібліотеку запускаємо

npm install @mui/material @emotion/react @emotion/styled

Якщо ми хоче му використовувати styled-components (https://www.npmjs.com/package//styled-components)

npm install @mui/material @mui/styled-engine-sc styled-components

Якщо хочете використовувати іконки то встановлюємо

npm install @mui/icons-material

Для того щоб використовувати CssBaseline

npm install @emotion/react @emotion/styled

CssBaseline

Компонент CssBaseline допомагає створити елегантну, узгоджену та просту базову стилізацію для компонентів та html. Підключається за допомогою компонента CssBaseline

```
import CssBaseline from "@mui/material/CssBaseline";

…

<CssBaseline />

…
```

Основан на нормалізації css https://github.com/necolas/normalize.css

Подробиці тут https://mui.com/material-ui/react-css-baseline/

Add components

Щоб використовувати компоненти з бібліотеки вантажимо їх до проєкту. Наприклад:

```
import Box from '@mui/material/Box';
```

Або

```
import { Box } from '@mui/material;
```

Деталі тут https://mui.com/material-ui/react-box/

Далі читаємо API копмонента і використовуємо атрібути. Наприклад:

```
<Box component="section"  sx={{ p: 2, border: '1px dashed grey' }}>
    This Box renders as an HTML section element.
</Box>
```

Деталі тут https://mui.com/material-ui/api/box/

Add components

Тут ви можете ознайомитись з усіма компонентами

https://mui.com/material-ui/getting-started/supported-components/

Customization

У вас є декілька варіантів для власної кастомізації будь якого компоненту:

1. Кастомізувати стилі компонента за допомогою атрібуна sx. Це є об’єкт стилізації як CSS. Наприклад

```
sx={{
    width: 300,
    color: 'success.main',
  }}
```

Щоб налаштувати певну частину компонента, ви можете використати ім’я класу, надане Material UI всередині sx prop.

```
sx={{
    width: 300,
    color: 'success.main',
    '& .MuiSlider-thumb': {
      borderRadius: '1px',
    },
  }}
```

2. Якщо ввам потрібно стилізуваи поведінку типу hover ви можете вокористовуваати класи MUI. Але ніколи не перезаписуйте стилі класи, робить це через спецефічність.

Подробиці тут https://mui.com/material-ui/customization/how-to-customize/

Customization

У вас є декілька варіантів для власної кастомізації будь якого компоненту:

1. Кастомізувати стилі компонента за допомогою атрібуна sx. Це є об’єкт стилізації як CSS. Наприклад

```
sx={{
    width: 300,
    color: 'success.main',
  }}
```

Щоб налаштувати певну частину компонента, ви можете використати ім’я класу, надане Material UI всередині sx prop.

```
sx={{
    width: 300,
    color: 'success.main',
    '& .MuiSlider-thumb': {
      borderRadius: '1px',
    },
  }}
```

Customization

2. Кастомізувати сам компонент та перевикористовувати його за допомогою утіліти styled(). Деталі тут https://mui.com/system/styled/. Наприклад:

```
const StyledButton = styled(Button)(({ theme }) => ({
  width: 300,
  color: theme.palette.success.main,
}));
export function MyButton({ children }) {
    return (
        <StyledButton>
            {children}
        </StyledButton>
)}
```

Customization

3. Створити власну тему. Для цього створюємо об’єкт теми за допомогою createTheme та передаємо йому новий об’єкт теми. Дефолтна конфігурація теми тут https://mui.com/material-ui/customization/default-theme/. Параметри кастомізації можна подивитись тут https://mui.com/material-ui/customization/theming/. Наприклад:

Для генерації теми використовйте:

https://zenoo.github.io/mui-theme-creator/

```
const theme : Theme = createTheme( options: {
    palette: {
        mode: 'dark',
        primary: {
            main: '#009cd8',
        },
        secondary: {
            main: '#f50057',
        },
        background: {
            default: '#333333',
            paper: '#111111',
        },
    },
});
```

Customization

Основні параметри теми та їх атрібути:

pallete - Палітра дозволяє змінювати колір компонентів відповідно до вашого бренду. Докладніше тут https://mui.com/material-ui/customization/palette/

typography - Набір розмірів тексту. Докладніше тут https://mui.com/material-ui/customization/typography/

spacing - встановлює послідовний інтервал між елементами вашого інтерфейсу. Докладніше тут https://mui.com/material-ui/customization/spacing/

breakpoints - API, що дозволяє використовувати точки зніни в різноманітних контекстах. Докладніше тут https://mui.com/material-ui/customization/breakpoints/

z-index - z-index — це властивість CSS, яка допомагає контролювати макет, надаючи третю вісь для впорядкування вмісту. Докладніше тут https://mui.com/material-ui/customization/z-index/

transition - дозволяють створювати користувальницькі переходи CSS, ви можете налаштовувати тривалість, ослаблення тощо. Докладніше тут https://mui.com/material-ui/customization/transitions/

Customization

Ви можете налаштувати стилі компонента, властивості за замовчуванням тощо, використовуючи ключ компонента всередині теми. Докладніше тут https://mui.com/material-ui/customization/theme-components/. Наприклад:

```
const theme = createTheme({
  components: {
    // Name of the component
    MuiButtonBase: {
      defaultProps: {
        // The props to change the default for.
        disableRipple: true,
      },
    },
  },
});
```

Customization

Далі вам потрібно передати новий об’єкт теми до ThemeProvider, та огорнути у провайдер компоненти. Наприклад:

```
<ThemeProvider  theme={theme}>
    <App/>
</ThemeProvider>
```

Dark mode

Material UI має два режими палітри: світлий (за замовчуванням) і темний.

Щоб використовувати режими потрібно:

1. Встановити mode у pallete теми.
2. Створити коруючий стан для mode.
3. Огорнути у useMemo() функцію toggleTheme.

Докладніше тут https://mui.com/material-ui/customization/dark-mode/

Formik

Formik — найпопулярніша у світі бібліотека форм із відкритим кодом для React і React Native. Він допоможе вам із 3 найбільш неприємними частинами при роботі з формами:

1. Отримання значень у стан форми та вихід із нього
2. Перевірка та повідомлення про помилки
3. Обробка подання форми

Сайт https://formik.org/

Документація https://formik.org/docs/overview

Приклад з використанням MUI https://formik.org/docs/examples/with-material-ui

Formik

Інсталяція

npm install formik –save

Formik відстежує стан вашої форми, а потім надає її разом із кількома повторно використовуваними методами та обробниками подій (handleChange, handleBlur і handleSubmit) у вашій формі за допомогою пропсів. handleChange і handleBlur працюють точно так, як очікувалося - вони використовують атрибут name або id, щоб визначити, яке поле потрібно оновити.

Formic надає вам набір компонентів для будування форми, а також набір хуків які допоможуть керувати формами.

Formik <Formik />

<Formik> — це компонент, який допомагає створювати форми. Він містить такі props:

- component - Може приймати базовий компонентз формою
- children - Дочірній контент.
- enableReinitialize - За замовчуванням false. Контролюйте, чи повинен Formik скидати форму, якщо початкові значення змінюються.
- initialErrors - Початкові помилки поля форми, Formik зробить ці значення доступними для відтворення компонента методів як помилки.
- initialStatus - Довільне значення для початкового стану форми. Якщо форму скинути, це значення буде відновлено.
- initialTouched - Початкові відвідані поля форми, Formik зробить ці значення доступними для візуалізації компонента методів, коли вони торкаються.
- initialValues - Початкові значення полів форми, Formik зробить ці значення доступними для відтворення компонента методів як значень. Навіть якщо ваша форма порожня за замовчуванням, ви повинні ініціалізувати всі поля початковими значеннями, інакше React видасть помилку про те, що ви змінили вхід з неконтрольованого на контрольований

Formik <Formik />

- onReset(values: Values, formikBag: FormikBag) - Ваш додатковий обробник скидання форми. Йому передаються значення ваших форм і "FormikBag".
- onSubmit: (values: Values, formikBag: FormikBag) => void | Promise<any> - Ваш обробник подання форми. Йому передаються значення ваших форм і «FormikBag», який містить об’єкт, що містить підмножину впроваджених властивостей і методів (тобто всі методи з іменами, що починаються з set<Thing> + resetForm) і будь-які властивості, які були передані в загорнутий компонент.
- validate?: (values: Values) => FormikErrors<Values> | Promise<any> - Перевірте значення форми за допомогою функції. Ця функція може бути: синхроною і повертає об’єкт помилок та асинхронною і повертає Promise, який вирішує об’єкт, що містить помилки.
- validateOnBlur - За умовчанням встановлено значення true. Використовуйте цей параметр, щоб запустити перевірку подій розмиття. Точніше, коли викликається handleBlur, setFieldTouched або setTouched.
- validateOnChange - За умовчанням встановлено значення true. Використовуйте цей параметр, щоб наказати Formik виконувати перевірку подій змін і методів, пов’язаних зі змінами. Точніше, коли викликається handleChange, setFieldValue або setValues.

Formik <Formik />

- validateOnMount - За замовчуванням false. Використовуйте цей параметр, щоб наказати Formik виконувати перевірки, коли компонент <Formik /> монтується та/або змінюються початкові значення.
- validationSchema?: Schema | (() => Schema) - Схема Yup (https://github.com/jquense/yup) або функція, яка повертає схему Yup. Це використовується для перевірки. Помилки зіставляються за допомогою ключа з помилками внутрішнього компонента. Його ключі мають збігатися з ключами значень.

Formik <Formik />

<Formik /> приймати функцію як дочірній контент і передавати туди такі параметри для рендорінгу:

- Dirty - Повертає true, якщо значення не дуже збігаються з початковими значеннями, інакше повертає false. dirty є обчислюваною властивістю лише для читання, і її не слід змінювати безпосередньо.
- Errors - Помилки перевірки форми. Має відповідати формі значень вашої форми, визначених у initialValues. Якщо ви використовуєте validationSchema (а це має бути), ключі та форма точно відповідатимуть вашій схемі. Внутрішньо Formik перетворює необроблені помилки перевірки Yup від вашого імені. Якщо ви використовуєте перевірку, тоді ця функція визначатиме форму об’єктів помилок.
- handleBlur: (e: any) => void - обробник події onBlur. Корисно, коли вам потрібно відстежити, чи було торкано введення чи ні. Це слід передати в <input onBlur={handleBlur} ... />
- handleChange: (e: React.ChangeEvent<any>) => void - Загальний обробник події зміни введення. Це оновить значення [ключ], де ключ — це атрибут назви вхідних даних, що видає подію. Якщо атрибут name відсутній, handleChange шукатиме атрибут id входу. Примітка: «введення» тут означає всі введення HTML.

Formik <Formik />

- handleReset: () => void - Скинути обробник. Поверне форму до початкового стану. Це слід передати до <button onClick={handleReset}>...</button>
- handleSubmit: (e: React.FormEvent<HTMLFormElement>) => void - Обробник подання. Це слід передати до <form onSubmit={props.handleSubmit}>...</form>. Детальніше тут https://formik.org/docs/guides/form-submission
- isSubmitting - Стан подання форми. Повертає true, якщо надсилання триває, і false в іншому випадку. ВАЖЛИВО: Formik встановить значення true, щойно буде зроблена спроба надсилання.
- IsValid - Повертає true, якщо немає помилок (тобто об'єкт errors порожній), і false в іншому випадку.
- IsValidating - Повертає true, якщо Formik запускає перевірку під час подання, або викликом [validateForm] безпосередньо false в іншому випадку.
- resetForm: (nextState?: Partial<FormikState<Values>>) => void - сброс форми. Якщо вказано nextState, Formik встановить nextState.values як новий «початковий стан» і використає відповідні значення nextState для оновлення початкових значень форми, а також initialTouched, initialStatus, initialErrors.

Formik <Formik />

- submitForm: () => Promise - Запустити надсилання форми. Обіцянку буде відхилено, якщо форма недійсна.
- setFieldValue: (field: string, value: any, shouldValidate?: boolean) => Promise<void | FormikErrors> - Встановіть значення поля імперативно. поле має відповідати ключу значень, які ви бажаєте оновити. Корисно для створення спеціальних обробників змін введення. Виклик цього призведе до запуску перевірки, якщо для validateOnChange встановлено значення true (що є за замовчуванням). Ви також можете явно заборонити/пропустити перевірку, передавши третій аргумент як false. Якщо для validateOnChange встановлено значення true і є помилки, їх буде вирішено у повернутому Promise.
- status - Об’єкт статусу верхнього рівня, який можна використовувати для представлення стану форми, який інакше не можна виразити/зберегти іншими методами. Це корисно для захоплення та передачі відповідей API вашому внутрішньому компоненту.
- touched: { [field: string]: boolean } - Торкнулися поля. Кожна клавіша відповідає полю, яке було торкано/відвідано.
- values: { [field: string]: any } - Значення вашої форми. Матиме форму результату mapPropsToValues (якщо вказано) або всіх реквізитів, які не є функціями, переданими вашому обернутому компоненту.

Formik <ErrorMessage />

<ErrorMessage /> — це компонент, який відображає повідомлення про помилку певного поля, якщо це поле було відвідано (тобто touched[name] === true) (і є повідомлення про помилку). Він очікує, що всі повідомлення про помилки зберігаються для заданого поля як рядок. Наприклад:

Formik <Form />

Form — це невелика обгортка навколо елемента HTML <form>, яка автоматично підключається до handleSubmit і handleReset Formik. Усі інші властивості передаються безпосередньо до вузла DOM.

Formik <Field />

<Field /> автоматично підключатиме input до Formik. Він використовує атрибут name для відповідності стану Formik. <Field /> за умовчанням буде елементом HTML <input />.

Props:

- as?: string | React.ComponentType<FieldProps['field']> - Або компонент React, або ім’я елемента HTML для рендерингу. Тобто одне з наступного:

input

select

textarea

HTML element

React custom component

Спеціальні компоненти React будуть передані onChange, onBlur, name та value, а також будь-які інші властивості, передані безпосередньо до <Field>.Типовим є 'input' (тому <input> відображається за умовчанням)

Formik <Field />

- children?: React.ReactNode | ((props: FieldProps) => React.ReactNode) - Або елементи JSX, або функція зворотного виклику. Те саме, що рендер.
- component?: string | React.ComponentType<FieldProps> - Або компонент React, або ім'я елемента HTML для рендерингу. Тобто одне з наступного:

input

select

textarea

A custom React component

- innerRef?: (el: React.HTMLElement<any> => void)
- name - Це обов'язкове поле для інпуту
- validate?: (value: any) => undefined | string | Promise<any> - Ви можете запустити незалежні перевірки на рівні поля, передавши функцію до властивості validate. Функція враховуватиме конфігурацію/реквізити validateOnBlur і validateOnChange, указані в батьківському елементі <Field> <Formik> / withFormik. Ця функція може бути як синхронною, так і асинхронною

Formik hooks

useFormik() - це спеціальний хук React, який безпосередньо повертатиме всі стани Formik і хелпери. Незважаючи на свою назву, він не призначений для більшості випадків використання. Всередині Formik використовує useFormik для створення компонента <Formik> (який рендерить React Context Provider). Якщо ви намагаєтеся отримати доступ до стану Formik через контекст, використовуйте useFormikContext. Використовуйте цей хук, лише якщо ви НЕ використовуєте <Formik> або withFormik.

useField() - це спеціальний хук React, який автоматично допоможе вам підключити вхідні дані до Formik. Ви можете і повинні використовувати його для створення власних примітивів введення.

Дякую за увагу