- **Publishing stories and reels:**

> **! Restrictions:**
> 1) We can`t get stories of other accounts in the current API implementation, only stories of our accounts with GET /{ig-user-id}/stories.
> 2) Currently, only business accounts can publish stories using the Content Publishing API.

For publishing stories, you need to use the same request as for media but add media_type=STORIES , and specify the path to the image or video in the image_url or video_url parameter.

For publishing reels, you need to add media_type=REELS.

About **user tags**, **product tags**, **collaborator tags**, and **creating a Carousel Container,** you can read more in the official documentation: https://developers.facebook.com/docs/instagram-api/guides/content-publishing.

Proceeding with all information, we assume that with some restrictions, we can create a personal assistant for us, but we have restrictions on interaction with other accounts (seeing their stories, putting likes).

The last calling question, what about messages? Can we chat with someone out of the Instagram application border?

In primary Instagram API, we can not do this, will investigate in the next article how we can cope with this by using Facebook Messanger API.

УДК 004.774

ВДОСКОНАЛЕННЯ СТВОРЕННЯ ФОРМ НА БАЗІ БІБЛІОТЕЦІ REACTJS

Гриценко К.М., студент магістратури механічної кафедри
Харківського національного автомобільно-дорожнього університету

Анотація: У статті розглянута актуальна проблема та необхідність вдосконалення роботи з динамічними формами на базі бібліотеці ReactJS, та як вдосконалити та зменшити час необхідний для розробки форм управління даними.

Ключові слова: ReactJS, форми, оптимізація, динамічне керування полями, залежності, шаблонні залежності.

Актуальність: На теперішній час ReactJS є широко використовуваною JavaScript-бібліотекою для створення клієнтських інтерфейсів. Особливо акту-

альним є його використання при розробці форм для введення та редагування даних, оскільки це одне з основних завдань у багатьох веб-додатках. При цьому неефективна розробка таких форм може призвести до погіршення досвіду користувача, зниження продуктивності додатків і навіть до вразливостей безпеки. Тому оптимізація форм розробки в ReactJS стає важливою для забезпечення високого рівня продуктивності, безпеки та зручності використання. Ключові аспекти оптимізації включають контрольовані компоненти, скорочення кількості перемальовок, обробку подій та перевірку введення даних, а також забезпечення доступу до даних лише через спеціальні функції доступу (акцесори). Ці заходи допомагають розробникам підвищити продуктивність своїх програм, надаючи більш стандартну, зручну та безпечну форму для роботи з даними.

Існує багато пакетів NPM для роботи з формами в ReactJS, які можуть значно облегшити процес розробки та підвищити ефективність кодування.

Наприклад, Formik - це популярний пакет, який пропонує спрощений спосіб управління формами в React. Він надає готові HOC і функціональні компоненти, які автоматизують управління станом форми, валідацію і відправку даних. Крім того, він підтримує інтеграцію з такими популярними бібліотеками, як Yup для перевірки форм і Redux для управління станом додатків.

Другий відомий пакет - це Final Form. Він також призначений для роботи з формами в React і пропонує подібний набір можливостей, як і Formik. Однак Final Form має кілька відмінностей, наприклад, він краще працює з великими формами завдяки своїй архітектурі, яка мінімізує кількість перерендерінгу.

React Hook Form - ще один популярний пакет, який дозволяє використовувати хуки React для роботи з формами. Він простий в освоєнні та пропонує легкий синтаксис для роботи з формами. Крім того, він має малий об'єм і не вимагає додаткових залежностей.

Усі три пакети мають активний розвиток і підтримують велику спільноту розробників, що робить їх хорошим вибором для роботи з формами в ReactJS. Вибір залежить від конкретних вимог проекту та особистих переваг розробника.

Дослідивши наявні пакети для роботи з формами було виявлено що немає пакета задля керування полями форми у залежності від введених даних.

Для роботи з даними часто потрібно керувати полями форми динамічно, наприклад:

- сховати поле якщо значення якогось поля у формі прийняло/не прийняло таке конкретне значення;
- показати поле якщо якесь поле у формі прийняло/не прийняло таке конкретне значення;
- дозволити/заборонити редагування поля у формі прийняло/не прийняло таке конкретне значення;

- встановити значення або очистити одного або багатьох полів, якщо інше поле у формі прийняло/не прийняло таке конкретне значення.

Перелічені завдання часто зустрічаються при створенні форм для роботи з даними.

Давайте розглянемо як можна реалізувати ці завдання, використовуючи популярні пакети для роботи з формами.

Оскільки пакети взяті за приклад однаково вирішують перелічені завдання, розглядатимемо лише один пакет, наприклад візьмемо пакет Formik.

Розглянемо конкретну імплементацію залежного поля з інших полів (рис. 1):

```
const MyField = (props) => {
  const {
    values: { textA, textB },
    touched,
    setFieldValue,
  } = useFormikContext();
  const [field, meta] = useField(props);

  React.useEffect(() => {
    // set the value of textC, based on textA and textB
    if (
      textA.trim() !== '' &&
      textB.trim() !== '' &&
      touched.textA &&
      touched.textB
    ) {
      setFieldValue(props.name, `textA: ${textA}, textB: ${textB}`);
    }
  }, [textB, textA, touched.textA, touched.textB, setFieldValue, props.name]);

  return (
    <>
      <input {...props} {...field} />
      {!!meta.touched && !!meta.error && <div>{meta.error}</div>}
    </>
  );
};
```

Рисунок 1 – Код, що реалізує залежність одного поля від двох інших

Таким чином реалізація залежного поля в кожному індивідуальному випадку описується кодом, що призводить до роздування коду та складності управління шаблонними залежностями, також при зміні будь-якої логіки або значень, необхідно залучати розробника для того, щоб змінити логіку роботи кожного окремого поля, що призводить до інкрементального ускладнення проекту та збільшення його вартості як у розробці так і за подальшої підтримки.

Метою мого дослідження на майбутнє є спрощення управління залежностями у формах за рахунок розробки шару абстракції роботи із залежностями та зручне управління поданням форми в іншому вигляді.

Висновок: реалізація динамічних залежностей між полями форми може призводити до ускладнення коду та збільшення часу на розробку та технічну підтримку. Проблема ускладнюється тим, що будь-яке редагування в логіці або параметрах вимагає втручання розробника для коригування коду кожного окремого поля. Метою майбутніх досліджень є розробка засобів, що спрощують управління залежностями у формах та надають більш зручний спосіб подання форми. Рекомендую використовувати популярні пакети, такі як Formik, React Hook Form або Final Form, щоб звести до мінімуму час на розробку та підтримку коду.

Література

1. ReactJs: Бібліотека для веб-інтерфейсу та нативного інтерфейсу користувача [Електронний ресурс]. Режим доступу: https://react.dev/. Дата доступу: 15.04.2024

2. Formik: world's most popular open source form library for React and React Native. [Електронний ресурс]. Режим доступу: https://formik.org/. Дата доступу: 15.04.2024

3. Final Form: Framework agnostic, high performance, subscription-based form state management. [Електронний ресурс]. Режим доступу: https://final-form.org/. Дата доступу: 15.04.2024

4. React Hook Form: Performant, flexible and extensible forms with easy-to-use validation. [Електронний ресурс]. Режим доступу: https://react-hook-form.com/. Дата доступу: 15.04.2024