ДОДАТОК А

КОД ПРОГРАМИ

Оснеовний компонент складу

```
import { NextPage } from 'next';
import { useQuery } from 'react-query';
import warehouse from '~/api/warehouse';
import { useRouter } from 'next/router';
import getBackendUrl from '~/utils/getBackendUrl';
import classNames from 'classnames';
import {
  Button,
  Intent,
} from '@blueprintjs/core';
import { useUser } from '~/contexts/user';
import { useCallback, useState } from 'react';
import { CreateOrderDialog } from
'~/components/screens/warehouse/CreateOrderDialog/CreateOrderDialog';
import { order } from '~/api/order';
import { productCount } from '~/api/product';
import { ActionTypes, actionCreator } from '~/utils/actionCreator';
import { ShipmentProduct } from './ShipmentProduct/ShipmentProduct';

const WarehousesPage: NextPage = () => {
  const router = useRouter();
  const { update, profileData, toasterRef } = useUser();
  const { data, refetch } = useQuery({
    queryKey: [ `warehouse_${ router.query.slug }`, profileData ],
    queryFn: async () => {
      const { slug } = router.query;
      if (slug) {
        try {
          const res = await warehouse.readByQuery({
            filter: {
              id: { _eq: slug },
            },
          });
          return res[0] || null;
        } catch (error) {
          return null;
        }
```

```
      }

      return null;
    },
  });

  const comeToWork = useCallback(() => {
    const currentValue = !!profileData?.at_warehouse;
    const currentWarehouse = !profileData?.current_warehouse ? data.id : null;
    update({
      at_warehouse: !currentValue,
      current_warehouse: currentWarehouse,
    });
  }, [ data, profileData, update ]);
  const buttonText = profileData?.at_warehouse ? 'Leave the warehouse' : 'Come to
work';
  const [ isOpenDialog, setIsOpenDialog ] = useState(false);
  const [ isOpenDialogShipment, setIsOpenDialogShipment ] = useState(false);
  const isCurrentWorcer = data?.workers.find((item) => item.id ===
profileData.id);
  if (!data) {
    return <div>Not found</div>;
  }

  return (
    <section className="container">
      <CreateOrderDialog
        refetchWarehouse={ refetch }
        warehouseId={ data.id }
        isOpenDialog={ isOpenDialog }
        setIsOpenDialog={ setIsOpenDialog }
      />
      <ShipmentProduct
        refetchWarehouse={ refetch }
        warehouseId={ data.id }
        isOpenDialog={ isOpenDialogShipment }
        setIsOpenDialog={ setIsOpenDialogShipment }
        warehouseItem={ data }
      />
```

```
    <div className="warhouse-page">
      <div className="flex gap-4">
        {isCurrentWorcer
      && <Button className="h-12" onClick={ comeToWork
}>{buttonText}</Button>}
        {isCurrentWorcer
      && <Button className="h-12" onClick={ () => { setIsOpenDialog(true); }
}>Create order</Button>}
        {isCurrentWorcer
      && <Button className="h-12" onClick={ () => {
setIsOpenDialogShipment(true); } }>Shipment of products</Button>}
      </div>

      <h1>{data.name}</h1>
      <div className="warhouse-page-item">
        <h2>Workers</h2>
        <table>
          <thead>
            <tr>
              <th>Image</th>
              <th>email</th>
              <th>last name</th>
              <th>first name</th>
              <th>at work</th>
            </tr>
          </thead>
          <tbody>
            {data.workers?.map((worker) => {
              return (
                <tr className="warhouse-item__block__item" key={ worker.id }>
                  <td><img src={ getBackendUrl(`/assets/${ worker.avatar }`) }
width={ 40 } height={ 40 } alt="" /></td>
                  <td>{worker.email}</td>
                  <td>{worker.last_name}</td>
                  <td>{worker.first_name}</td>
                  <td>
                    <div className={ classNames('at-work', {
                      'at-work--true': worker.at_warehouse && `${
worker.current_warehouse }` === router.query.slug,
```

```
                }) }
                >
                </div>
              </td>
            </tr>
          );
        })}
      </tbody>

    </table>
  </div>
  <div className="warhouse-page-item">
    <h2>Products</h2>
    <table>
      <thead>
        <tr>
          <th>Image</th>
          <th>Count</th>
          <th>Name</th>

        </tr>
      </thead>
      <tbody>
        {data.product?.map((product__count) => {
          const { product } = product__count;

          if (!product__count.count) {
            return null;
          }
          return (
            <tr className="warhouse-item__block__item" key={ product.id }>
              <td><img src={ getBackendUrl(`/assets/${ product.image }`) }
width={ 40 } height={ 40 } alt="" /></td>
              <td>{product__count.count}</td>

              <td>{product.name}</td>
            </tr>
          );
```

```
          })}
        </tbody>

      </table>
    </div>
    {data.order.length ? (
      <div className="warhouse-page-item">
        <div className="flex gap-4 items-center mb-3">
          <h2>Orders</h2>

        </div>

        {data.order?.map((orderItem) => {
          const { product_count, id, description } = orderItem;
          return (
            <div className="warhouse-page-item" key={ orderItem.id }>
              <div className="flex gap-4 items-center mb-3">
                <h3>
                  Order number:
                  {' '}
                  {id}
                </h3>
                <h3>{description}</h3>
                <Button
                  type="button"
                  onClick={ async () => {
                    const productsInWarehouse = [ ...data.product ];
                    orderItem.product_count.forEach(async (element) => {
                      const currentItem = data.product
                        .find((a) => a.product.id === element.product.id);
                      if (currentItem) {
                        currentItem.count += element.count;
                        await productCount.updateOne(currentItem.id, { count:
currentItem.count });

                        return;
                      }

                      await warehouse.updateOne(
```

```
                    data.id,
                    { product_count: [ ...productsInWarehouse.map((b) => b.id),
element.id ] },
                   );
                   productsInWarehouse.push(element);
                  });

                  const newOrder = await order.updateOne(orderItem.id, { is_accept:
true });
                  const result = await warehouse.updateOne(
                   data.id,
                   { order: data.order.filter((c) => c.id !== orderItem.id) },
                  );

                  actionCreator({
                   actionType: ActionTypes.accept_order, warehouseId: data.id,
orderId: orderItem.id, userId: profileData.id,
                  });

                  if (toasterRef?.current) {
                   toasterRef.current.show({
                    message: 'Your order has been accepted',
                    timeout: 3000,
                    intent: Intent.SUCCESS,
                   });
                  }
                  refetch();
                 } }
                >
                 Accept order
                </Button>
               </div>
               <table>
                <thead>
                 <tr>
                  <th>Image</th>
                  <th>Count</th>

                  <th>Name</th>
```

```
                  </tr>
                </thead>
                <tbody>
                  {product_count.map((product_count_item) => {
                    return (
                      <tr key={ product_count_item.id } className="warhouse-
item__block__item" key={ product_count_item?.id }>
                        <td><img src={ getBackendUrl(`/assets/${
product_count_item?.product.image }`) } width={ 40 } height={ 40 } alt=""
/></td>
                        <td>{product_count_item?.count}</td>
                        <td>{product_count_item?.name}</td>

    </tr>
);
})}
</tbody>
</table>
</div>

);
})}
</div>
):null}

</div>
</section>
);
};

exportdefaultWarehousesPage;
```

Компонент модального вікна створення замовлення

```
import React, { useMemo, useState } from 'react';
import { ItemPredicate, ItemRenderer, Select } from '@blueprintjs/select';
import getBackendUrl from '~/utils/getBackendUrl';
import product, { productCount } from '~/api/product';
import { useQuery } from 'react-query';
```

```
import { IProduct } from '~/interfaces/IProduct';
import { uniqueId } from '@blueprintjs/core/lib/esm/common/utils';
import classNames from 'classnames';
import { useUser } from '~/contexts/user';
import { order } from '~/api/order';
import {
  MenuItem, Button, Dialog, DialogBody, DialogFooter,
  FormGroup,
  InputGroup,
  Intent,
} from '@blueprintjs/core';
import { ActionTypes, actionCreator } from '~/utils/actionCreator';

export interface ProductType {
  title: string;
  year: number;
  rank: number;
}

export const filterProduct: ItemPredicate<IProduct> = (query, film, _index,
exactMatch) => {
  const normalizedTitle = film.name?.toLowerCase();
  const normalizedQuery = query.toLowerCase();

  if (exactMatch) {
    return normalizedTitle === normalizedQuery;
  }
  return `${ normalizedTitle }`.indexOf(normalizedQuery) >= 0;
};

export const renderProducts: ItemRenderer<IProduct> = (film, {
  handleClick, handleFocus, modifiers, query,
}) => {
  if (!modifiers.matchesPredicate) {
    return null;
  }
  return (
    <MenuItem
      active={ modifiers.active }
```

```
      disabled={ modifiers.disabled }
      key={ film.rank }
      labelElement={ (
        <img src={ getBackendUrl(`/assets/${ film.image }`) } width={ 32 }
height={ 32 } alt="" />
) }
      onClick={ handleClick }
      onFocus={ handleFocus }
      roleStructure="listoption"
      text={ `${ film.name }` }
    />
  );
};

export type PropsProductSelect = {
  items: any[],
  setSelectProducts: React.Dispatch<React.SetStateAction<Record<string,
number>>>,
  selectId: string,
  onChange: (e: IProduct) => void
  className: string
};
export const ProductSelect: React.FC<PropsProductSelect> = ({
  items, setSelectProducts, selectId, onChange, className,
}) => {
  const [ selectedFilm, setSelectedFilm ] = useState<IProduct | undefined>();
  return (
    <Select<IProduct>
      className={ className }
      items={ items }
      itemPredicate={ filterProduct }
      itemRenderer={ renderProducts }
      noResults={ <MenuItem disabled text="No results." roleStructure="listoption"
/> }
      onItemSelect={ (e) => {
        setSelectProducts((prev) => {
          return {
            ...prev,
            [selectId]: e.id,
```

```
        };
      });
      setSelectedFilm(e);
      onChange(e);
    } }
  >
    <Button
      // text={ selectedFilm?.name || 'Select product' }
      rightIcon="double-caret-vertical"
      placeholder="Select a film"
      className="create-order-input"
    >
      {selectedFilm && (
      <img
        src={ getBackendUrl(`/assets/${ selectedFilm.image }`) }
        width={ 20 }
        height={ 20 }
        alt=""
      />
      )}
      {selectedFilm?.name || 'Select product'}
    </Button>
  </Select>
 );
};

export const CreateOrderDialog = ({
 warehouseId, setIsOpenDialog, isOpenDialog, refetchWarehouse = () => {},
}) => {
 const { data: productsFromApi } = useQuery<IProduct[] | null>({
  queryKey: [ 'products' ],
  queryFn: async () => {
   try {
    const res = await product.readByQuery({});
    return res || null;
   } catch (error) {
    return null;
   }
  },
```

```
  });
  const [ selectProducts, setSelectProducts ] = useState<Record<string,
number>>({});
  const renderProducts: IProduct[] = useMemo(() => {
    if (productsFromApi) {
      return productsFromApi?.filter((item) =>
!Object.values(selectProducts).includes(item.id));
    }

    return [];
  }, [ productsFromApi, selectProducts ]);
  const [ inputValues, setInputValues ] = useState<{
    id: string;
    product: IProduct | null;
    value: string;
  }[]>([ {
    id: 'select1',
    product: null,
    value: '',
  } ]);
  const [ errors, setErrors ] = useState<string[]>([]);
  const { toasterRef } = useUser();
  return (
    <Dialog
      isOpen={ isOpenDialog }
      onClose={ () => { setIsOpenDialog(false); } }
      title="Create order"
      icon="info-sign"
    >
      <DialogBody>
        {inputValues.map((item) => {
          return (
            <FormGroup key={ item.id } className="form-group">
              <ProductSelect
                selectId={ item.id }
                items={ renderProducts }
                setSelectProducts={ setSelectProducts }
                className={ classNames({
```

```
            error: errors.includes(item.id)
              && !inputValues.find((valueItem) => valueItem.id ===
item.id)?.product,

          }) }
          onChange={ (e: IProduct) => {
            setInputValues((prev) => prev.map((valueItem) => {
              if (valueItem.id === item.id) {
                return {
                  ...valueItem,
                  product: e,
                };
              }

              return valueItem;
            }));
            setErrors((prev) => {
              return prev.filter((errorsItem) => errorsItem !== item.id);
            });
          } }
        />
        <InputGroup
          onChange={ (e) => {
            setInputValues((prev) => prev.map((valueItem) => {
              if (valueItem.id === item.id) {
                return {
                  ...valueItem,
                  value: e.target.value,
                };
              }

              return valueItem;
            }));
            setErrors((prev) => {
              return prev.filter((errorsItem) => errorsItem !== item.id);
            });
          } }
          placeholder="Number"
          type="number"
```

```
            className={ classNames({
              error: errors.includes(item.id)
                && !inputValues.find((valueItem) => valueItem.id ===
item.id)?.value,

            }) }
           />
           <Button onClick={ () => {
            setInputValues((prev) => prev.filter((valueItem) => valueItem.id !==
item.id));
            setSelectProducts((prev) => {
              const newResult = { ...prev };
              console.log(item.id);
              delete newResult[item.id];

              return newResult;
            });
           } }
           >
            Delete
           </Button>
          </FormGroup>
        );
       })}
       <Button
        disabled={ !renderProducts.length || productsFromApi?.length ===
inputValues.length }
        onClick={ () => {
         setInputValues((prev) => [ ...prev, {
          id: uniqueId('select'),
          product: null,
          value: '',
         } ]);
        } }
       >
        Add product
       </Button>
      </DialogBody>
      <DialogFooter
```

```
        actions={ (
          <>
            <Button
              text="Close"
              onClick={ () => {
                setIsOpenDialog(false);
              } }
            />
            <Button
              intent="primary"
              disabled={ !inputValues.length }
              text="Create order"
              onClick={ async () => {
                const retirnValue = inputValues
                  .filter((item) => !item.value || !item.product)
                  .map((item) => item.id);
                setErrors(retirnValue);

                if (retirnValue.length === 0) {
                  try {
                    const { data: newProducts } = await
productCount.createMany(inputValues
                      .map((item) => ({ product: item.product?.id, count: +item.value })));

                    const orderItem = await order.createOne({
                      product_count: newProducts.map((item) => item.id),
                      warehouse: warehouseId,
                    });

                    refetchWarehouse();

                    actionCreator({
                      actionType: ActionTypes.create_order,
                      warehouseId,
                      orderId: orderItem.id,
                    });
                    setInputValues([ {
                      id: 'select1',
                      product: null,
```

```
                value: '',
              } ]);
              setSelectProducts({});

              if (toasterRef?.current) {
                toasterRef.current.show({
                  message: 'Your order has been created',
                  timeout: 3000,
                  intent: Intent.SUCCESS,
                });
              }
              setIsOpenDialog(false);
            } catch (error) {
              if (toasterRef?.current) {
                toasterRef.current.show({
                  message: 'Create error',
                  timeout: 3000,
                  intent: Intent.DANGER,
                });
              }
            }
          }
        }
      />
    </>
  ) }
  />
 </Dialog>
 );
};
```

ДОДАТОК Б

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

Формат	Позиція.	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кільк.	Примітки
			Текстові документи		
А4	1	ГЮІК.505900.06 ПЗ	Пояснювальна записка	1	с.
			Інші документи		
А4	2		Код програми	1	с.
А4	3		Демонстраційний матеріал	1	с.

Змін.	Арк	Номер докум.	Підпис	Дата	ГЮИК. 505900.06 ВД			
Розроб.		Науменко Д. І.			Розроблення системи автоматизованого управління складською системою з використання хмарних технологій	Літера	Аркуш	Аркуші
Перевір		Бронніков А.І.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр		Демська Н. П.						
Затв.		Невлюдов І.Ш.						