**REACT.JS**

Table of Contents:

    Reacts Fragments..................................................................................1

    React Portals......................................................................................... 1

    React Refs (References).........................................................................2

    What is an Effect?..................................................................................3

    React use Reducer () .............................................................................3

    React use State () ..................................................................................4

Project ………………………………………………………………………………………………5

Outputs……………………………………………………………………………………………...18

1.React Fragments:

* We can use <> and </> instead of <div> and </div>. Because, it is an empty wrapper component. It doesn’t render any real HTML element to the DOM (Document Object Model). But it fulfills React’s/JSX requirement.
* The Code Execution is faster with use of React Fragments as Compared to the div tag.
* Less utilization of memory with the use of react Fragments instead of div tag.
* You can declare Fragments in two types: -
* <> and </>
* <Fragment> and </Fragment>

2. React Portals:

* React Portal is a first-class way to render child components into a DOM node outside of the parent DOM hierarchy defined by the component hierarchy.

3. React Refs (References):

* Refs are a function provided by react to access the DOM element and react element that you might have created on your own.
* They are used in cases where we want to change the value of the child component, without using props and all.
* They also provide us with good functionality as we can use callbacks with them.
* Depending on the type of the node, the value of the ref can defer.
* The ref attribute doesn’t have instances, and can’t be used on Functional Components.

Example: Adding Refs to DOM elements.

**import** React, { Component } from 'react';     
import { render } from 'react-dom';     
class App **extends** React.Component {     
constructor(props) {     
super(props);     
this.callRef = React.createRef();     
this.addingRefInput = **this**.addingRefInput.bind(**this**);     
}     
addingRefInput() {     
this.callRef.current.focus();     
}     
render() {     
return (     
<div>     
<h1>Hello World</h1>     
<input     
type="text"     
ref={**this**.callRef} />     
<input     
type="button"     
value="Enter text"     
onClick={**this**.addingRefInput}     
/>     
</div>    
);     
}     
}     
export **default** App;

* **Output:**   
  

4. React use Effect ():

* The use Effect () hook allows you to performs side effects in your components.
* Some examples of side components are: fetching data, directly updating the DOM and timers.

5. React use Reducer ():

* The use Reducer () can be used as a replacement for use State (), if you need “more powerful state management”.
* It allows custom state logic.
* The use Reducer () hook accepts two arguments.

* How is use Reducer () is used?

Const [state, dispatch FN] = use Reducer (reducer FN, initial State, init FN);

* State= The state snapshot used in the component render
* Dispatch FN= A function that is used to dispatch a new action
* Reducer FN= A function that is triggered automatically once an action is dispatched.
* Intial State= Initial state
* Init FN= A function to set the initial state programmatically.

6. React use State ():

* The main state management tool
* Great for independent use of state.
* Great if state updates are easy and limited to a few kinds of updates.
* At the top of your component, import the use State () Hook:

import {use State} from "react";

7. Started the first task to create new fields in the user list.

To Start my first task, I first clone the repository folder into my vs code

* Then I import react-simple2 folder into frontend vscode.
* Then entered npm install. Now install node-modules with below code
* **Npm i --force**
* Then I entered the following syntax to get work of npm:
* **Npm start**
* This above command starts the app in development mode.
* Now I get UI Login Page in server.
* Now I import maven project as React-src folder into backend eclipse
* Now connect to sql and creating nproduts database in sql and connect to database
* Now run the java application in backend

Now login to the UI: -

* First add the new fields in the create user list like phone number and join date.
* Now add the calendar Icon in join date field.
* Now click on create user we get that successfully user created.

Code: -

UserTableRow.tsx: -

import { useState } from 'react';

// @mui

import {

  Stack,

  Avatar,

  Button,

  Checkbox,

  TableRow,

  MenuItem,

  TableCell,

  IconButton,

  Typography,

  TextField,

} from '@mui/material';

// @types

import { IUser, IUserAccountGeneral } from '../../../../@types/user';

// components

import Label from '../../../../components/label';

import Iconify from '../../../../components/iconify';

import MenuPopover from '../../../../components/menu-popover';

import ConfirmDialog from '../../../../components/confirm-dialog';

// ----------------------------------------------------------------------

type Props = {

  row: IUser;

  selected: boolean;

  onEditRow: VoidFunction;

  onSelectRow: VoidFunction;

  onDeleteRow: VoidFunction;

};

export default function UserTableRow({

  row,

  selected,

  onEditRow,

  onSelectRow,

  onDeleteRow,

}: Props) {

  const { firstName, lastName, email, role, **PhoneNumber, JoinDate**,  } = row;

  const [openConfirm, setOpenConfirm] = useState(false);

  const [openPopover, setOpenPopover] = useState<HTMLElement | null>(null);

  const handleOpenConfirm = () => {

    setOpenConfirm(true);

  };

  const handleCloseConfirm = () => {

    setOpenConfirm(false);

  };

  const handleOpenPopover = (event: React.MouseEvent<HTMLElement>) => {

    setOpenPopover(event.currentTarget);

  };

  const handleClosePopover = () => {

    setOpenPopover(null);

  };

  return (

    <>

      <TableRow hover selected={selected}>

        <TableCell padding="checkbox">

          <Checkbox checked={selected} onClick={onSelectRow} />

        </TableCell>

        <TableCell>

          <Stack direction="row" alignItems="center" spacing={2}>

            <Typography variant="subtitle2" noWrap>

              {firstName}{' '}

              {lastName}

            </Typography>

          </Stack>

        </TableCell>

        <TableCell align="left">{email}</TableCell>

        <TableCell align="left" sx={{ textTransform: 'capitalize' }}>

          {role}</TableCell>

**<TableCell align="left">{PhoneNumber}</TableCell>**

**<TableCell align="left">{JoinDate} </TableCell>**

        <TableCell align="right">

          <IconButton color={openPopover ? 'inherit' : 'default'} onClick={handleOpenPopover}>

            <Iconify icon="eva:more-vertical-fill" />

          </IconButton>

        </TableCell>

      </TableRow>

      <MenuPopover

        open={openPopover}

        onClose={handleClosePopover}

        arrow="right-top"

        sx={{ width: 140 }}

      >

        <MenuItem

          onClick={() => {

            handleOpenConfirm();

            handleClosePopover();

          }}

          sx={{ color: 'error.main' }}

        >

          <Iconify icon="eva:trash-2-outline" />

          Delete

        </MenuItem>

        <MenuItem

          onClick={() => {

            onEditRow();

            handleClosePopover();

          }}

        >

          <Iconify icon="eva:edit-fill" />

          Edit

        </MenuItem>

      </MenuPopover>

      <ConfirmDialog

        open={openConfirm}

        onClose={handleCloseConfirm}

        title="Delete"

        content="Are you sure want to delete?"

        action={

          <Button variant="contained" color="error" onClick={onDeleteRow}>

            Delete

          </Button>

        }

      />

    </>

  );

}

UserNewEditForm.tsx: -

import \* as Yup from 'yup';

import { useEffect, useMemo } from 'react';

import { useNavigate } from 'react-router-dom';

// form

import { Controller, useForm } from 'react-hook-form';

import { yupResolver } from '@hookform/resolvers/yup';

// @mui

import {  LoadingButton } from '@mui/lab';

import { Box, Card, Grid, Stack, TextField, TextFieldProps } from '@mui/material';

// utils

// routes

import { dispatch } from 'src/redux/store';

import { addUser, updateUser } from 'src/redux/slices/user';

import { PATH\_DASHBOARD } from '../../../routes/paths';

// @types

import { IUser } from '../../../@types/user';

// assets

// components

import { CustomFile } from '../../../components/upload';

import { useSnackbar } from '../../../components/snackbar';

import FormProvider, { RHFTextField } from '../../../components/hook-form';

import { DatePicker } from '@mui/x-date-pickers/DatePicker';

// ----------------------------------------------------------------------

interface FormValuesProps extends Omit<IUser, 'avatarUrl'> {

  avatarUrl: CustomFile | string | null;

}

type Props = {

  isEdit?: boolean;

  currentUser?: IUser;

};

export default function UserNewEditForm({ isEdit = false, currentUser }: Props) {

  const navigate = useNavigate();

  const { enqueueSnackbar } = useSnackbar();

  const NewUserSchema = Yup.object().shape({

    firstName: Yup.string().required('Name is required'),

    lastName: Yup.string().required('Name is required'),

    email: Yup.string().required('Email is required').email('Email must be a valid email address'),

    role: Yup.string().required('Role is required'),

**PhoneNumber: Yup.string().required('Number is requried'),**

**JoinDate: Yup.string().required('Date is requried'),**

  });

  const defaultValues = useMemo(

    () => ({

      firstName: currentUser?.firstName || '',

      lastName: currentUser?.lastName || '',

      email: currentUser?.email || '',

      role: currentUser?.role || '',

**phonenumber: currentUser?.PhoneNumber || '',**

**JoinDate: currentUser?.JoinDate || '',**

    }),

    // eslint-disable-next-line react-hooks/exhaustive-deps

    [currentUser]

  );

  const methods = useForm<FormValuesProps>({

    resolver: yupResolver(NewUserSchema),

    defaultValues,

  });

  const {

    reset,

    watch,

    control,

    setValue,

    handleSubmit,

    formState: { isSubmitting },

  } = methods;

  const values = watch();

  useEffect(() => {

    if (isEdit && currentUser) {

      reset(defaultValues);

    }

    if (!isEdit) {

      reset(defaultValues);

    }

    // eslint-disable-next-line react-hooks/exhaustive-deps

  }, [isEdit, currentUser]);

  const onSubmit = async (data: FormValuesProps) => {

    console.log('in submit');

    const request: IUser = {

      id: data.id,

      firstName: data.firstName,

      lastName: data.lastName,

      email: data.email,

      role: data.role,

**PhoneNumber: data.PhoneNumber,**

**JoinDate: data.JoinDate,**

    };

    try {

      if (isEdit && currentUser) {

        request.id = data.id;

        dispatch(updateUser(request));

      }

      if (!isEdit) {

        dispatch(addUser(request));

        reset();

      }

      enqueueSnackbar(!isEdit ? 'Create success!' : 'Update success!');

      navigate(PATH\_DASHBOARD.user.list);

      console.log('DATA', data);

    } catch (error) {

      console.error(error);

    }

  };

  return (

    <FormProvider methods={methods} onSubmit={handleSubmit(onSubmit)}>

      <Grid container spacing={3}>

        <Grid item xs={12} md={8}>

          <Card sx={{ p: 3 }}>

            <Box

              rowGap={3}

              columnGap={2}

              display="grid"

              gridTemplateColumns={{

                xs: 'repeat(1, 1fr)',

                sm: 'repeat(2, 1fr)',

              }}

            >

              <RHFTextField name="firstName" label="First Name" />

              <RHFTextField name="lastName" label="Last Name" />

              <RHFTextField name="email" label="Email Address" />

              <RHFTextField name="role" label="Role" />

**<RHFTextField name="PhoneNumber" label="Phone Number" />**

**{/\* <RHFTextField name="JoinDate" label="Join Date" /> \*/}**

**<Controller**

**name="JoinDate"**

**control={control}**

**render={({ field, fieldState: { error } }) => (**

**<DatePicker**

**label="JoinDate"**

**value={field.value}**

**onChange={(newValue: any) => {**

**field.onChange(newValue);**

**}}**

**renderInput={(params: JSX.IntrinsicAttributes & TextFieldProps) => (**

**<TextField**

**{...params}**

**fullWidth**

**error={!!error}**

**helperText={error?.message}**

**/>**

**)}**

**/>**

**)}**

**/>**

            </Box>

            <Stack alignItems="flex-end" sx={{ mt: 3 }}>

              <LoadingButton type="submit" variant="contained" loading={isSubmitting}>

                {!isEdit ? 'Create User' : 'Save Changes'}

              </LoadingButton>

            </Stack>

          </Card>

        </Grid>

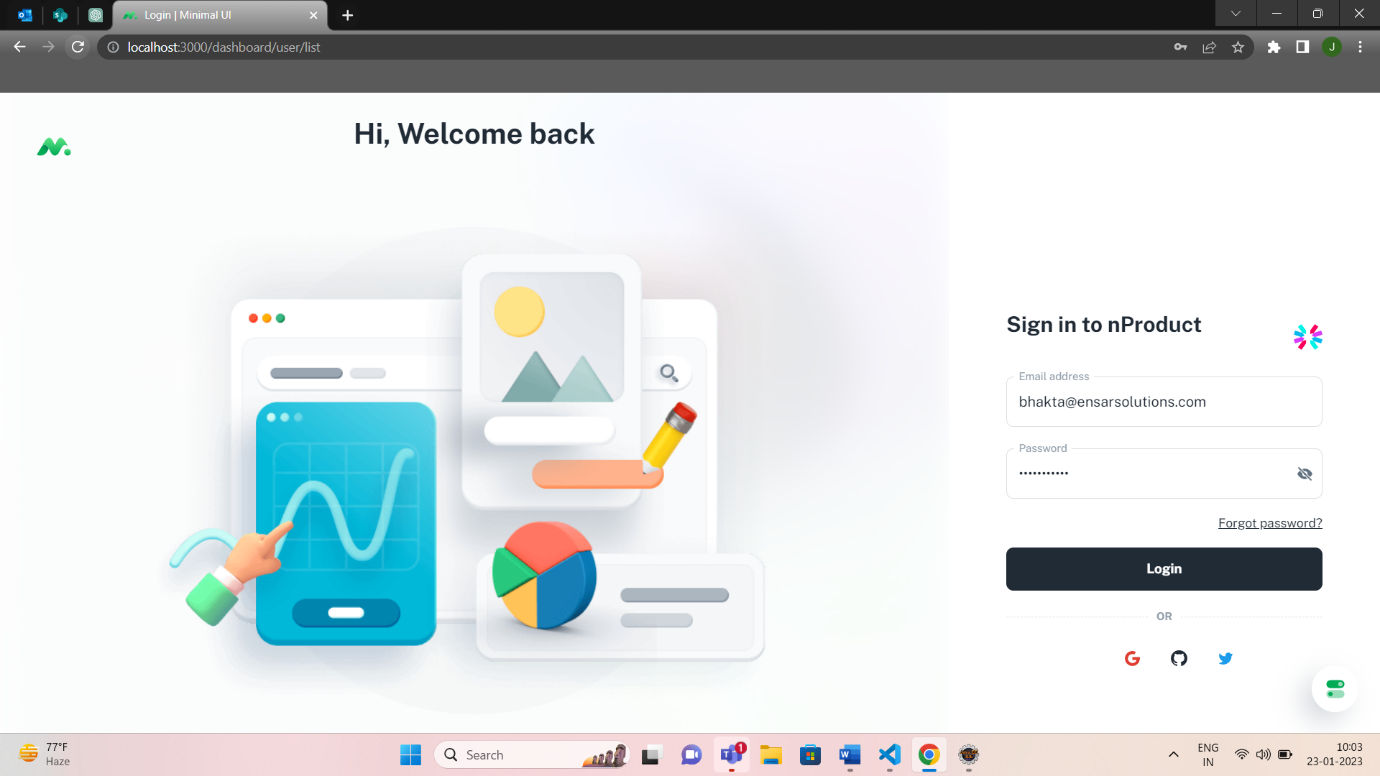
      </Grid>

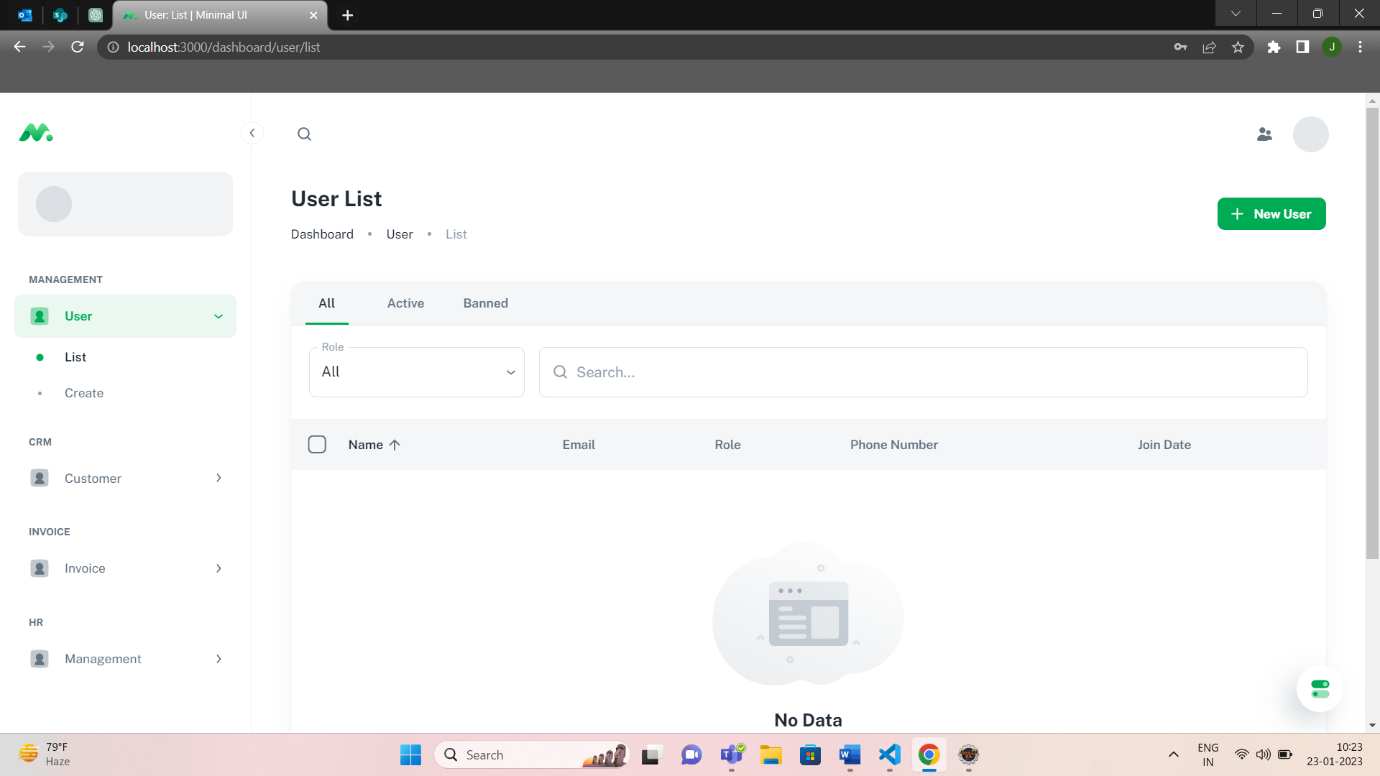
    </FormProvider>

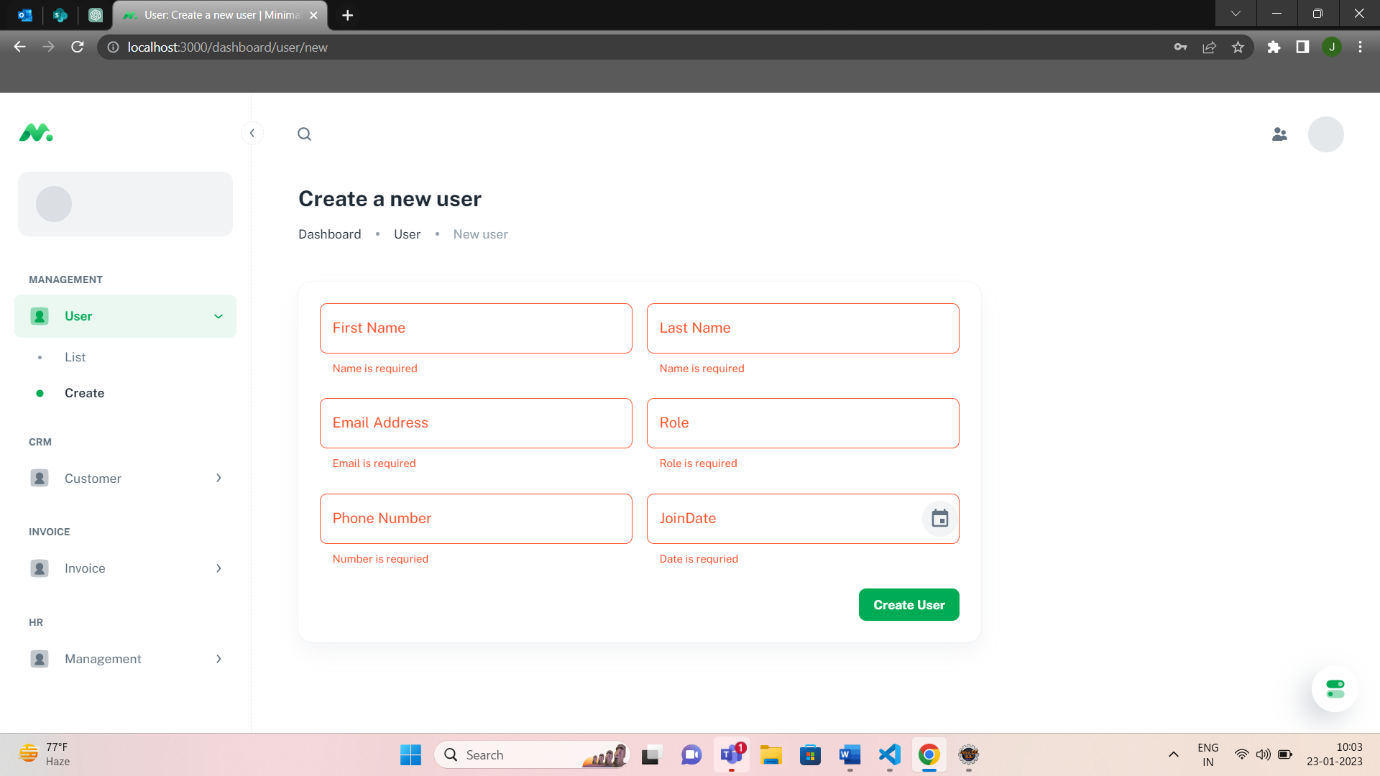
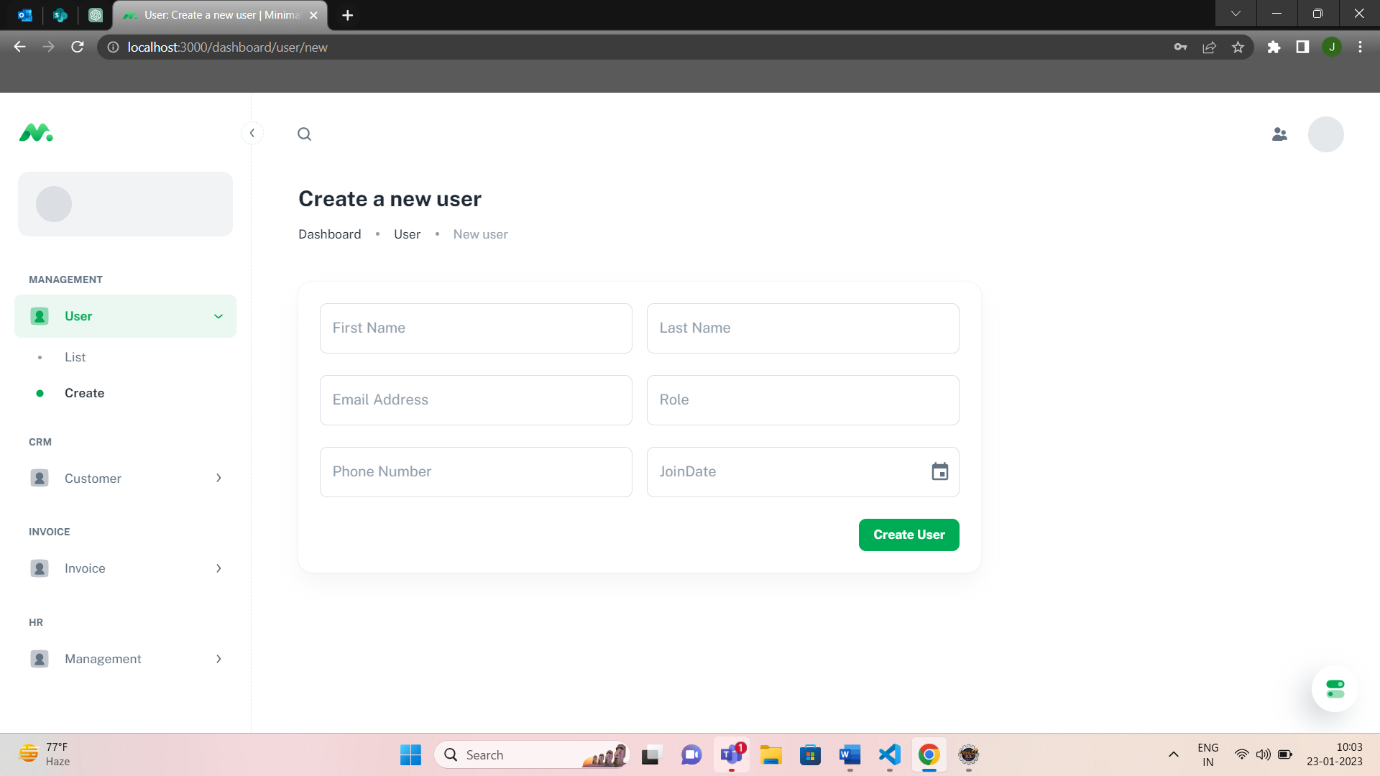
  );

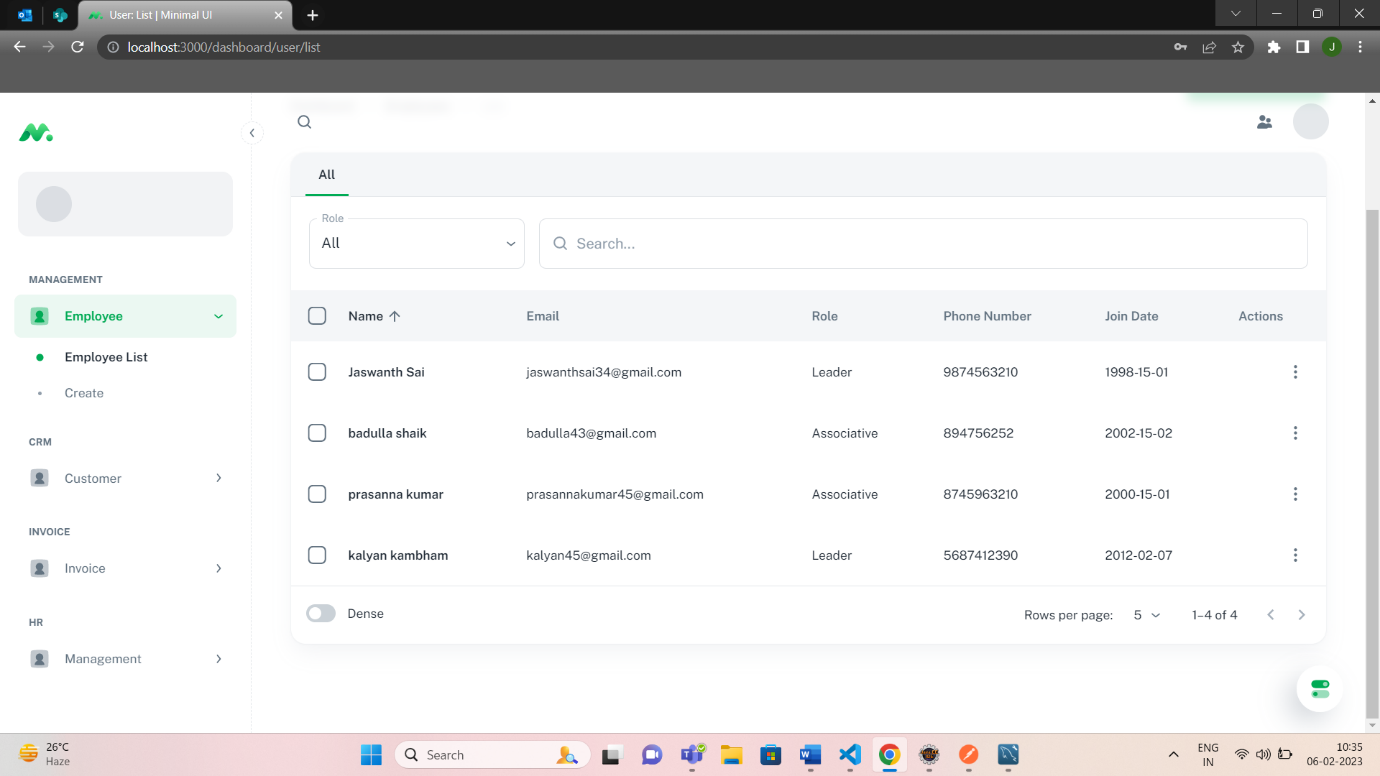
}

OUTPUTS: -









*Thank You.*