**Tutorial สำหรับปฏิบัติการ 2**

สร้าง directory Project ชื่อ sa-65-example

                cd c:\

mkdirsa-65-example

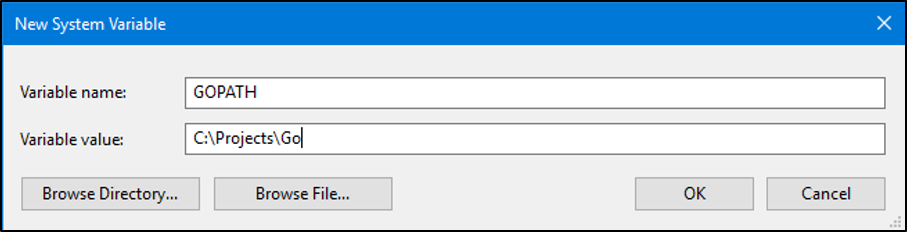
                cd sa-65-example

1. ติดตั้ง Go compiler โดย download ได้จากที่นี่<https://golang.org/dl/>

รูปภาพประกอบด้วย ข้อความ

คำอธิบายที่สร้างโดยอัตโนมัติ

ตั้งค่า Environment ดังนี้



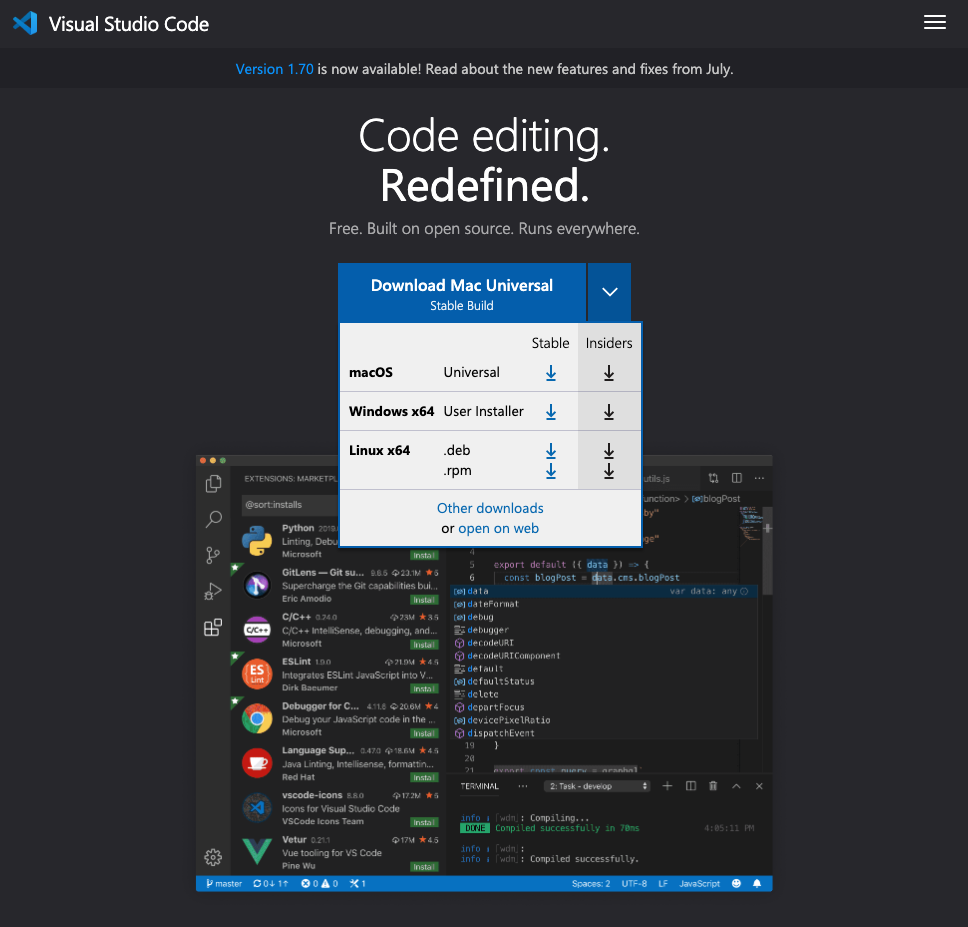
GOROOT ตั้งเป็น c:\Go

GOPATH ตั้งเป็น c:\Users\<ชื่อ user ของตนเอง>\Go

PATH เพิ่ม c:\Go\bin;c:\Users\<ชื่อ user ของตนเอง>\Go\bin; เข้าไปด้านหน้า

ถ้าเปิด Command Prompt (cmd.exe) หรือ Power Shell อยู่ ให้ปิดแล้วเปิดใหม่

1. ติดตั้ง VS Code



เลือก Extension ติดตั้ง extension สำหรับภาษา Go

1. ติดตั้ง nodejs (เราจะใช้ v16.x ในเทอม 1/65)

รูปภาพประกอบด้วย ข้อความ

คำอธิบายที่สร้างโดยอัตโนมัติ

1. สมัคร Github (https://github.com) แล้วจะได้ชื่อ user ของ GitHub มา

1. Install Git<https://git-scm.com/download/win>

1. สร้าง project backend ที่ c:\sa-65-example

cd c:\sa-65-example

mkdir backend

cd backend

go mod init **github.com/<ชื่อ user ที่ได้มาจากการสมัคร GitHub>/sa-65-example**

เช่น ของอาจารย์จะเป็น github.com/chanwit/sa-65-example

1. ติดตั้ง GORM (ถ้าเรียก go get ไม่ได้แปลว่าการติดตั้ง Go มีปัญหา)

go get -u github.com/gin-gonic/gin

go get -u gorm.io/gorm

go get -u gorm.io/driver/sqlite

mkdir entity

cd entity/

1. ใน c:\sa-65-example\backend\entity สร้าง schema ของ User ในไฟล์ชื่อ user.go

แก้ไขไฟล์ user.go ให้เป็นแบบนี้

**package entity**

**import (**

**"time"**

**"gorm.io/gorm"**

**)**

**type User struct {**

**gorm.Model**

**FirstName        string**

**LastName         string**

**Email            string**

**Age              uint8**

**BirthDay         time.Time**

**}**

จากตำแหน่ง folder เดียวกัน เราจะเตรียมไฟล์ถัดมาคือไฟล์ setup.go เพื่อใช้สร้าง database

**package entity**

**import (**

**"gorm.io/gorm"**

**"gorm.io/driver/sqlite"**

**)**

**var db \*gorm.DB**

**func DB() \*gorm.DB {**

**return db**

**}**

**func SetupDatabase() {**

**database, err := gorm.Open(sqlite.Open("sa-65.db"), &gorm.Config{})**

**if err != nil {**

**panic("failed to connect database")**

**}**

**// Migrate the schema**

**database.AutoMigrate(&User{})**

**db = database**

**}**

เมื่อได้ Entity แล้ว เราจะเตรียมส่วนถัดมาคือ controller

ขั้นตอนการสร้าง controller คือ ที่ folder c:\sa-65-example\backend

1. mkdir controller
2. cd controller

1. ใน c:\sa-65-example\backend\controller สร้างไฟล์ชื่อ user.go เพื่อเก็บ controller สำหรับเชื่อมต่อกับ entity User

**package controller**

**import (**

**"github.com/<ชื่อ github id ของตนเอง>/sa-65-example/entity"**

**"github.com/gin-gonic/gin"**

**"net/http"**

**)**

จากนั้นสร้าง function ต่อไปนี้

1. function CreateUser เป็นการทำงานแทนคำสั่ง insert ของ SQL

**// POST /users**

**func CreateUser(c \*gin.Context) {**

**var user entity.User**

**if err := c.ShouldBindJSON(&user); err != nil {**

**c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})**

**return**

**}**

**if err := entity.DB().Create(&user).Error; err != nil {**

**c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})**

**return**

**}**

**c.JSON(http.StatusOK, gin.H{"data": user})**

**}**

โดย function นี้จะคืนค่าเป็น user ที่สร้างเสร็จแล้ว กลับไปเป็น JSON ให้ฝั่ง UI นำไปแสดงผล

ถัดมาจะเป็น function GetUser โดยในตัวอย่างเป็นการตั้งใจใช้คำสั่ง SELECT … WHERE id =... เพื่อดึงข้อมูล user ออกมาตาม primary key ที่กำหนด ผ่าน func DB.Raw(...)

**// GET /user/:id**

**func GetUser(c \*gin.Context) {**

**var user entity.User**

**id := c.Param("id")**

**if err := entity.DB().Raw("SELECT \* FROM users WHERE id = ?", id).Scan(&user).Error; err != nil {**

**c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})**

**return**

**}**

**c.JSON(http.StatusOK, gin.H{"data": user})**

**}**

function ถัดมา ListUsers จะเป็นการ list รายการของ User ออกมา โดยแสดงการใช้ SELECT \*

ผลลัพธ์ที่เป็น รายการข้อมูลจะสามารถดึงออกมาได้อย่างถูกต้องเมื่อนำตัวแปรที่เป็น array มารับ ในตัวอย่างนี้ users เป็นตัวแปรประเภท array ของ entity.User (สังเกต []entity.User)

**// GET /users**

**func ListUsers(c \*gin.Context) {**

**var users []entity.User**

**if err := entity.DB().Raw("SELECT \* FROM users").Scan(&users).Error; err != nil {**

**c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})**

**return**

**}**

**c.JSON(http.StatusOK, gin.H{"data": users})**

**}**

function ถัดมาเป็น function สำหรับลบ user ด้วย ID ก็คือการ DELETE … WHERE ID=...

**// DELETE /users/:id**

**func DeleteUser(c \*gin.Context) {**

**id := c.Param("id")**

**if tx := entity.DB().Exec("DELETE FROM users WHERE id = ?", id); tx.RowsAffected == 0 {**

**c.JSON(http.StatusBadRequest, gin.H{"error": "user not found"})**

**return**

**}**

**c.JSON(http.StatusOK, gin.H{"data": id})**

**}**

และ function สุดท้ายคือ function สำหรับ update user ก็คือการ UPDATE … WHERE ID=... ในตัวอย่างใช้คำสั่ง DB.Save() แทน update ของ SQL

**// PATCH /users**

**func UpdateUser(c \*gin.Context) {**

**var user entity.User**

**if err := c.ShouldBindJSON(&user); err != nil {**

**c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})**

**return**

**}**

**if tx := entity.DB().Where("id = ?", user.ID).First(&user); tx.RowsAffected == 0 {**

**c.JSON(http.StatusBadRequest, gin.H{"error": "user not found"})**

**return**

**}**

**if err := entity.DB().Save(&user).Error; err != nil {**

**c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})**

**return**

**}**

**c.JSON(http.StatusOK, gin.H{"data": user})**

**}**

เมื่อเราเขียน function ทั้งหมดที่เป็น CRUD แล้วก็นำมาประกาศเป็น path ของ API ด้วย router ในไฟล์ main.go ซึ่งจะอยู่ใน directory นอกสุด (ตำแหน่งเดียวกับที่ go.mod อยู่)

สร้างไฟล์ main.go และเขียนโปรแกรมสำหรับสร้าง Server ดังต่อไปนี้

อย่าลืมแก้ github.com/chanwit/sa-65-example เป็นชื่ออื่น ๆ ที่ระบุไว้ใน go.mod และใน user.go

**package main**

**import (**

**"github.com/chanwit/sa-65-example/controller"**

**"github.com/chanwit/sa-65-example/entity"**

**"github.com/gin-gonic/gin"**

**)**

**func main() {**

**entity.SetupDatabase()**

**r := gin.Default()**

**// User Routes**

**r.GET("/users", controller.ListUsers)**

**r.GET("/user/:id", controller.GetUser)**

**r.POST("/users", controller.CreateUser)**

**r.PATCH("/users", controller.UpdateUser)**

**r.DELETE("/users/:id", controller.DeleteUser)**

**// Run the server**

**r.Run()**

**}**

สั่ง download dependency ด้วยคำสั่ง

go mod tidy

แล้วสั่ง compile โปรแกรม backend ด้วยคำสั่ง

go build -o main.exe main.go

หรือ

go build -o main main.go บน Linux และ macOS

ถ้า error ให้ลง GCC \*\* exec: "gcc": executable file not found in %PATH%

ติดตั้ง GCC compiler:<https://github.com/jmeubank/tdm-gcc/releases/download/v9.2.0-tdm64-1/tdm64-gcc-9.2.0.exe>

เมื่อโปรแกรม compile ได้อย่างถูกต้องแล้ว

รันโปรแกรมโดยการรันคำสั่ง main

.\main.exe

หรือ

./main บน Linux และ macOS

**Frontend**

update version ของ npm ให้เป็นรุ่นล่าสุด

npm install -g npm@latest

Install : yarn

<https://classic.yarnpkg.com/en/docs/install/#windows-stable>

1. สร้างโปรเจค react

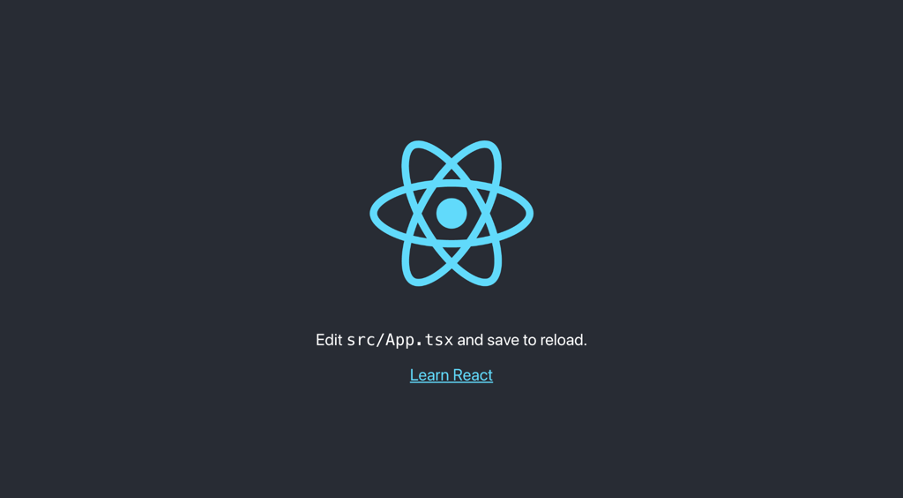
                ป้อนคำว่า  **frontend** เป็นชื่อ app

npx create-react-app <<ชื่อโปรเจค>> --template typescript

cd <<ชื่อโปรเจค>>

npm start

1. เปิด Browser เพื่อทดสอบผลการ run<http://localhost:3000/>



1. ติดตั้ง **Package** ที่ต้องใช้งาน

ติดตั้ง package สำหรับจัดการ Router

npm install --save @types/react-router-dom

npm install --save react-router-dom@6.x

1. ติดตั้ง **Material-UI** (<https://material-ui.com/getting-started/installation/> )

สำหรับใช้งาน component ต่างๆ ของ material เช่น button, box, grid เป็นต้น

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

หรือ

yarn add @mui/material @emotion/react @emotion/styled

สำหรับใช้งาน icon

npm install @mui/icons-material หรือ yarn add @mui/icons-material

สำหรับใช้งาน date picker

npm i @mui/x-date-pickers หรือ yarn add @mui/x-date-pickers

npm i @date-io/date-fns

สำหรับใช้งาน Data Table

npm i @mui/x-data-grid หรือ yarn add @mui/x-data-grid

สำหรับจัดการ Date & Time

npm install date-fns

npm install moment

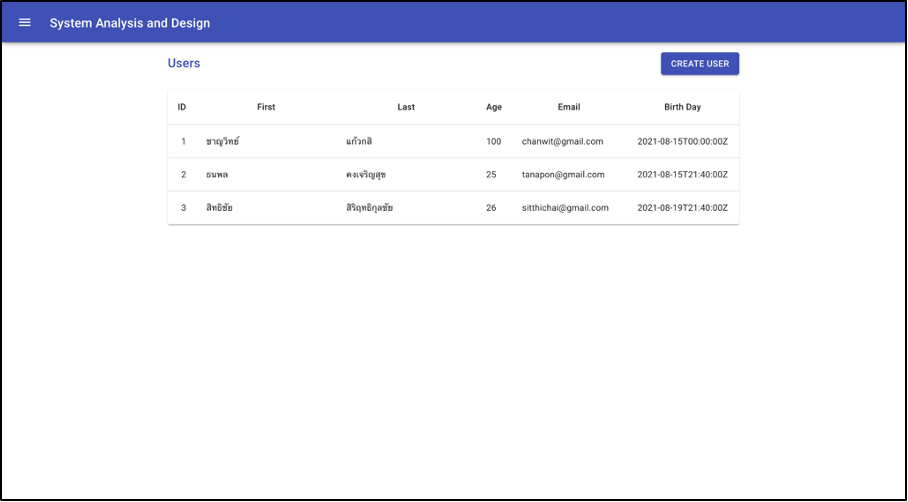
1. ปรับปรุง source code frontend เพื่อทดสอบการเชื่อมต่อกับ backend สำหรับส่งข้อมูล และรับข้อมูล

หน้าตาเว็บที่เราจะทำเป็นดังนี้

-           หน้าแรกเป็นหน้า แสดงข้อมูลผู้ใช้ ที่ GET ข้อมูลจาก Database

-           หน้าสำหรับ Insert ข้อมูล User เข้าไปที่ Database

หน้าแรก



หน้าเพิ่มข้อมูล User

รูปภาพประกอบด้วย ข้อความ

คำอธิบายที่สร้างโดยอัตโนมัติ

เมื่อเห็นภาพของระบบแล้วเราก็จะเริ่ม การแก้ไข source code ของโปรแกรมเรา

โครงสร้างไฟล์ที่เราต้องสร้างใน directory frontend

./frontend

        /...

        /src

                    /...

                    /App.tsx

                    /models

                        /IUser.ts

                    /components

                        /Navbar.tsx

                        /Users.tsx

                        /UserCreate.tsx

-           เริ่มต้นด้วยการจัดการ Router โดยการแก้ไขไฟล์ **App.tsx** จะอยู่ใน directory ชื่อ src

import React from "react";

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import Navbar from "./components/Navbar";

import Users from "./components/Users";

import UserCreate from "./components/UserCreate";

export default function App() {

return (

  <Router>

   <div>

   <Navbar />

   <Routes>

       <Route path="/" element={<Users />} />

       <Route path="/create" element={<UserCreate />} />

   </Routes>

   </div>

  </Router>

);

}

จะเห็นว่ามี path="/" เรียกใช้ component User ที่ import มาจาก./components/Users คือหน้าแสดงข้อมูล user

และ path="/create" เรียกใช้ component UserCreate ที่ import มาจาก./components/UserCreate คือหน้าสร้างข้อมูล user

                ส่วนของ Navbar เรา import เข้ามาเพื่อใช้ทุกๆหน้า

-           สร้าง User Interface เพื่อประกาศโครงสร้างข้อมูลของ User ว่ามี field ชื่ออะไร และ ชนิดข้อมูลเป็นอะไร

สร้าง directory ชื่อ models ใน directory  src จากนั้นสร้างไฟล์​ **IUser.ts**

export interface UsersInterface {

 ID: string,

 FirstName: string;

 LastName: string;

 Email: string;

 Age: number;

 BirthDay: Date | null;

}

-           สร้าง directory ชื่อ components ใน directory  src และสร้างไฟล์ต่อไปนี้

สร้างไฟล์ **Navbar.tsx**ดังนี้

import \* as React from "react";

import AppBar from "@mui/material/AppBar";

import Box from "@mui/material/Box";

import Toolbar from "@mui/material/Toolbar";

import Typography from "@mui/material/Typography";

import IconButton from "@mui/material/IconButton";

import MenuIcon from "@mui/icons-material/Menu";

function Navbar() {

 return (

   <Box sx={{ flexGrow: 1 }}>

     <AppBar position="static">

       <Toolbar>

         <IconButton

           size="large"

           edge="start"

           color="inherit"

           aria-label="menu"

           sx={{ mr: 2 }}

         >

           <MenuIcon />

         </IconButton>

         <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>

           System Analysis and Design

         </Typography>

       </Toolbar>

     </AppBar>

   </Box>

 );

}

export default Navbar;

สร้างไฟล์ **Users.tsx**ดังนี้

import React, { useEffect } from "react";

import { Link as RouterLink } from "react-router-dom";

import Typography from "@mui/material/Typography";

import Button from "@mui/material/Button";

import Container from "@mui/material/Container";

import Box from "@mui/material/Box";

import { UsersInterface } from "../models/IUser";

import { DataGrid, GridColDef } from "@mui/x-data-grid";

function Users() {

 const [users, setUsers] = React.useState<UsersInterface[]>([]);

 const getUsers = async () => {

   const apiUrl = "http://localhost:8080/users";

   const requestOptions = {

     method: "GET",

     headers: { "Content-Type": "application/json" },

   };

   fetch(apiUrl, requestOptions)

     .then((response) => response.json())

     .then((res) => {

       console.log(res.data);

       if (res.data) {

         setUsers(res.data);

       }

     });

 };

 const columns: GridColDef[] = [

   { field: "ID", headerName: "ID", width: 50 },

   { field: "FirstName", headerName: "First name", width: 150 },

   { field: "LastName", headerName: "Last name", width: 150 },

   { field: "Email", headerName: "Email", width: 200 },

   { field: "Age", headerName: "Age", width: 100 },

   { field: "BirthDay", headerName: "BirthDay", width: 200 },

 ];

 useEffect(() => {

   getUsers();

 }, []);

 return (

   <div>

     <Container maxWidth="md">

       <Box

         display="flex"

         sx={{

           marginTop: 2,

         }}

       >

         <Box flexGrow={1}>

           <Typography

             component="h2"

             variant="h6"

             color="primary"

             gutterBottom

           >

             Users

           </Typography>

         </Box>

         <Box>

           <Button

             component={RouterLink}

             to="/create"

             variant="contained"

             color="primary"

           >

             Create User

           </Button>

         </Box>

       </Box>

       <div style={{ height: 400, width: "100%", marginTop: '20px'}}>

         <DataGrid

           rows={users}

           getRowId={(row) => row.ID}

           columns={columns}

           pageSize={5}

           rowsPerPageOptions={[5]}

         />

       </div>

     </Container>

   </div>

 );

}

export default Users;

สร้างไฟล์ **UserCreate.tsx**ดังนี้

import React from "react";

import { Link as RouterLink } from "react-router-dom";

import TextField from "@mui/material/TextField";

import Button from "@mui/material/Button";

import FormControl from "@mui/material/FormControl";

import Container from "@mui/material/Container";

import Paper from "@mui/material/Paper";

import Grid from "@mui/material/Grid";

import Box from "@mui/material/Box";

import Typography from "@mui/material/Typography";

import Divider from "@mui/material/Divider";

import Snackbar from "@mui/material/Snackbar";

import MuiAlert, { AlertProps } from "@mui/material/Alert";

import { UsersInterface } from "../models/IUser";

import { AdapterDateFns } from "@mui/x-date-pickers/AdapterDateFns";

import { LocalizationProvider } from "@mui/x-date-pickers/LocalizationProvider";

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

const Alert = React.forwardRef<HTMLDivElement, AlertProps>(function Alert(

 props,

 ref

) {

 return <MuiAlert elevation={6} ref={ref} variant="filled" {...props} />;

});

function UserCreate() {

 const [date, setDate] = React.useState<Date | null>(null);

 const [user, setUser] = React.useState<Partial<UsersInterface>>({});

 const [success, setSuccess] = React.useState(false);

 const [error, setError] = React.useState(false);

 const handleClose = (

   event?: React.SyntheticEvent | Event,

   reason?: string

 ) => {

   if (reason === "clickaway") {

     return;

   }

   setSuccess(false);

   setError(false);

 };

 const handleInputChange = (

   event: React.ChangeEvent<{ id?: string; value: any }>

 ) => {

   const id = event.target.id as keyof typeof UserCreate;

   const { value } = event.target;

   setUser({ ...user, [id]: value });

 };

 function submit() {

   let data = {

     FirstName: user.FirstName ?? "",

     LastName: user.LastName ?? "",

     Email: user.Email ?? "",

     Age: typeof user.Age === "string" ? parseInt(user.Age) : 0,

     BirthDay: date,

   };

   const apiUrl = "http://localhost:8080/users";

   const requestOptions = {

     method: "POST",

     headers: { "Content-Type": "application/json" },

     body: JSON.stringify(data),

   };

   fetch(apiUrl, requestOptions)

     .then((response) => response.json())

     .then((res) => {

       if (res.data) {

         setSuccess(true);

       } else {

         setError(true);

       }

     });

 }

 return (

   <Container maxWidth="md">

     <Snackbar

       open={success}

       autoHideDuration={6000}

       onClose={handleClose}

       anchorOrigin={{ vertical: "bottom", horizontal: "center" }}

     >

       <Alert onClose={handleClose} severity="success">

         บันทึกข้อมูลสำเร็จ

       </Alert>

     </Snackbar>

     <Snackbar open={error} autoHideDuration={6000} onClose={handleClose}>

       <Alert onClose={handleClose} severity="error">

         บันทึกข้อมูลไม่สำเร็จ

       </Alert>

     </Snackbar>

     <Paper>

       <Box

         display="flex"

         sx={{

           marginTop: 2,

         }}

       >

         <Box sx={{ paddingX: 2, paddingY: 1 }}>

           <Typography

             component="h2"

             variant="h6"

             color="primary"

             gutterBottom

           >

             Create User

           </Typography>

         </Box>

       </Box>

       <Divider />

       <Grid container spacing={3} sx={{ padding: 2 }}>

         <Grid item xs={6}>

           <p>First Name</p>

           <FormControl fullWidth variant="outlined">

             <TextField

               id="FirstName"

               variant="outlined"

               type="string"

               size="medium"

               value={user.FirstName || ""}

               onChange={handleInputChange}

             />

           </FormControl>

         </Grid>

         <Grid item xs={6}>

           <FormControl fullWidth variant="outlined">

             <p>Last Name</p>

             <TextField

               id="LastName"

               variant="outlined"

               type="string"

               size="medium"

               value={user.LastName || ""}

               onChange={handleInputChange}

             />

           </FormControl>

         </Grid>

         <Grid item xs={12}>

           <FormControl fullWidth variant="outlined">

             <p>Email</p>

             <TextField

               id="Email"

               variant="outlined"

               type="string"

               size="medium"

               value={user.Email || ""}

               onChange={handleInputChange}

             />

           </FormControl>

         </Grid>

         <Grid item xs={6}>

           <FormControl fullWidth variant="outlined">

             <p>Age</p>

             <TextField

               id="Age"

               variant="outlined"

               type="number"

               size="medium"

               InputProps={{ inputProps: { min: 1 } }}

               InputLabelProps={{

                 shrink: true,

               }}

               value={user.Age || ""}

               onChange={handleInputChange}

             />

           </FormControl>

         </Grid>

         <Grid item xs={6}>

           <FormControl fullWidth variant="outlined">

             <p>BirthDay</p>

             <LocalizationProvider dateAdapter={AdapterDateFns}>

               <DatePicker

                 value={date}

                 onChange={(newValue) => {

                   setDate(newValue);

                 }}

                 renderInput={(params) => <TextField {...params} />}

               />

             </LocalizationProvider>

           </FormControl>

         </Grid>

         <Grid item xs={12}>

           <Button component={RouterLink} to="/" variant="contained">

             Back

           </Button>

           <Button

             style={{ float: "right" }}

             onClick={submit}

             variant="contained"

             color="primary"

           >

             Submit

           </Button>

         </Grid>

       </Grid>

     </Paper>

   </Container>

 );

}

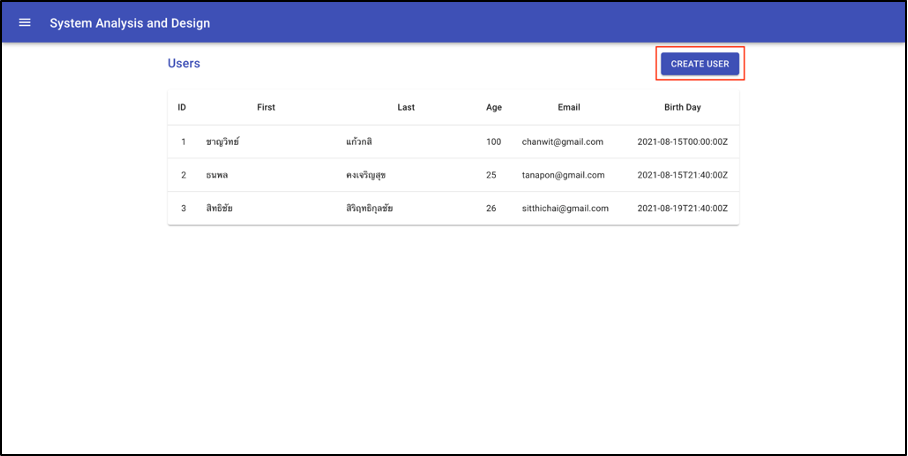
export default UserCreate;

1. สั่ง npm start ที่ directory frontend เพื่อ start frontend

กดปุ่ม Create User เพิ่อเพิ่มข้อมูล User จากนั้นกลับมาที่หน้าหลัก ก็จะพบกับข้อมูลที่เพิ่มเข้าไป \*\*\*\*อย่าลืม  run backend\*\*\*\*

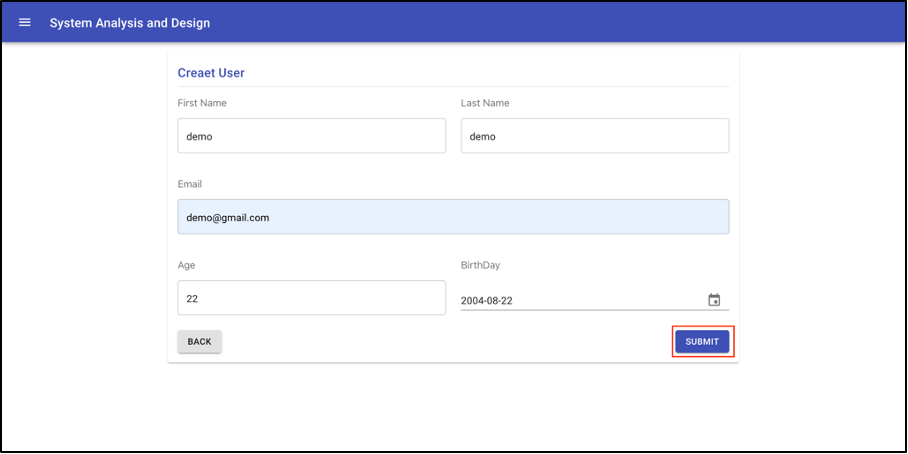
1. ทดสอบเพิ่มข้อมูล และแสดงผลข้อมูล

เปิดเว็บเบราว์เซอร์ที่[http://localhost:3000](http://localhost:3000/)

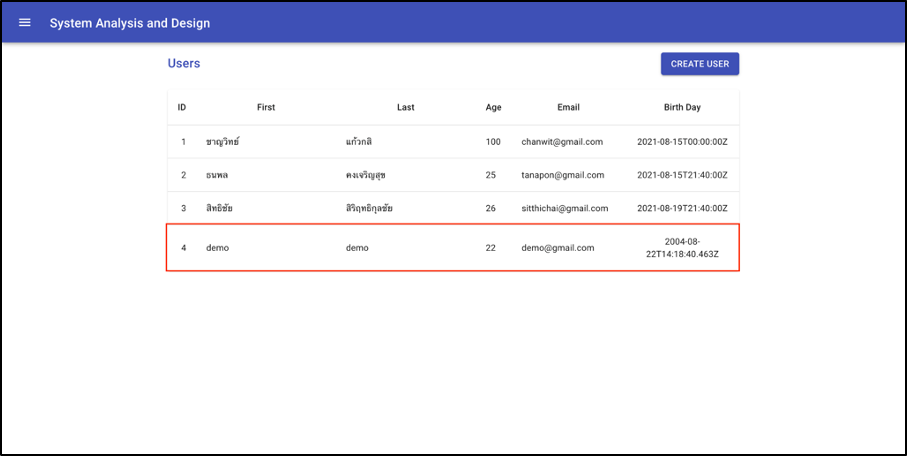


\* หมายเหตุ : ข้อมูลที่แสดงเป็นข้อมูลจากการ create user

ให้ดำเนินการเพิ่มข้อมูล โดยกดที่ปุ่ม create user -> กรอกข้อมูลให้ครบถ้วนจากนั้นกด submit



แสดงผลข้อมูลที่เพิ่ม กดปุ่ม back เพื่อกลับไปหน้าแรก



1. กรณีติด Access-Control-Allow-Origin เพิ่มข้อมูลไม่ได้

ให้เพิ่มฟังก์ชัน CORSMiddleware() ในไฟล์ main.go ของ backend

func CORSMiddleware() gin.HandlerFunc {

  return func(c \*gin.Context) {

        c.Writer.Header().Set("Access-Control-Allow-Origin", "\*")

        c.Writer.Header().Set("Access-Control-Allow-Credentials", "true")

        c.Writer.Header().Set("Access-Control-Allow-Headers", "Content-Type, Content-Length, Accept-Encoding, X-CSRF-Token, Authorization, accept, origin, Cache-Control, X-Requested-With")

        c.Writer.Header().Set("Access-Control-Allow-Methods", "POST, OPTIONS, GET, PUT")

        if c.Request.Method == "OPTIONS" {

          c.AbortWithStatus(204)

          return

        }

        c.Next()

  }

}

และเรียกใช้ CORSMiddleware() ในฟังก์ชั่น main เรียกใช้ก่อน Routes

func main() {

...

r.Use(CORSMiddleware())

...

}

สั่ง go build -o main.exe main.go ใหม่อีกครั้ง

และ run backend ใหม่ ด้วยคำสั่ง

.\main.exe

หรือ

./main บน Linux และ macOS

------------------------------------- จบ -------------------------------------