МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

КАТЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Інженерія програмного забезпечення

Лабораторна робота №9

«Проєктування та реалізація програмного продукту з використанням архітектурної моделі «Клієнт-Сервер»»

Виконав:
студент групи ІО-32
Крадожон М.
Перевірив(-ла):
Васильєва М.

Київ – 2023

Лабораторна робота №9

Тема: Проєктування та реалізація програмного продукту з використанням архітектурної моделі «Клієнт-Сервер»

Мета: Вивчити реалізацію та специфіку використання архітектури «клієнт-сервер». Спроєктувати програмну систему (або її частину), архітектура якої відповідала б програмній моделі «Клієнт-Сервер».

Варіант: 3215 % 19 = 4

Система управління замовленнями ресторану. Розробити систему керування замовленнями та доставкою їжі для ресторану. Клієнти можуть переглядати меню, робити замовлення через веб-сайт. Персонал ресторану може приймати та обробляти замовлення, встановлювати статуси готовності та відстежувати доставку через серверну частину системи.

Tech Stack:

Back-End:

- Основою серверної частини стала мова програмування **JavaScript**, у поєднанні із середовищем виконання **Node.js**.
- Для створення REST API застосовано популярний фреймворк **Express.js**.

Front-End:

- Для клієнтської частини використано бібліотеку **React**, яка забезпечила створення динамічного та інтерактивного інтерфейсу.
- Задля оптимізації продуктивності та реалізації серверного рендерингу використано фреймворк **Next.js**.
- Для стилізації інтерфейсу залучено CSS-фреймворк **TailwindCSS**, що забезпечує зручну і швидку роботу зі стилями.
- У створенні інтерфейсів також використовувались компоненти бібліотеки **shadcn/ui**.

Управління станом і даними:

- Використано бібліотеку **TanStack Query**, яка спростила процес управління запитами до сервера і синхронізацію даних.
- Для роботи з формами залучено бібліотеку **React Hook Form**, що забезпечила ефективну перевірку та обробку введених даних.

HTTP-запити:

- Для обміну даними між клієнтом і сервером було застосовано **Axios**, що дозволило здійснювати запити у зручному і зрозумілому форматі.

Код (JavaScript & TypeScript):

Back-end

index.js

```
const express = require('express')
const cors = require('cors')
const bodyParser = require('body-parser')
const path = require('path')

const { MenuManager, OrderManager } = require('./classes')

// Constants
const PORT = 5000
const app = express()

// File paths
const menuFilePath = path.join(__dirname, './data/menu.json')
const ordersFilePath = path.join(__dirname, './data/orders.json')

// Instances
const menuManager = new MenuManager(menuFilePath)
const orderManager = new OrderManager(ordersFilePath)

// Middleware
app.use(bodyParser.json())
app.use(cors())

// Routes
app.get('/menu', (_, res) => {
const action = menuManager.getAllMenuItems()

res.json(action)
 })

 app.post('/menu', (req, res) => {
const newMenuItem = req.body
const action = menuManager.addMenuItem(newMenuItem)

res.status(201).json({ message: 'Menu item added', item: action })
 })

 app.get('/menu/:id', (req, res) => {
const { id } = req.params
const action = menuManager.getMenuItemById(id)

if (!action) {
 res.status(404).json({ message: 'Menu item not found' })
} else {
 res.json(action)
}
```

```
})

app.get('/orders', (_, res) => {
const action = orderManager.getAllOrders()

res.json(action)
})

app.post('/orders', (req, res) => {
const newOrder = req.body
const action = orderManager.addOrder(newOrder)

res.status(201).json({ message: 'Order created', order: action })
})

app.get('/orders/:id', (req, res) => {
const { id } = req.params
const action = orderManager.getOrderById(id)

if (!action) {
 res.status(404).json({ message: 'Order not found' })
} else {
 res.json(action)
}
})

app.patch('/orders/:id', (req, res) => {
const { id } = req.params
const { status } = req.body
const action = orderManager.updateOrderStatus(id, status)

if (!action) {
 res.status(404).json({ message: 'Order not found' })
} else {
 res.json({ message: 'Order status updated', order: action })
}
})

app.delete('/orders/:id', (req, res) => {
const { id } = req.params
const action = orderManager.deleteOrder(id)

if (!action) {
 res.status(404).json({ message: 'Order not found' })
} else {
 res.json({ message: 'Order deleted', order: action })
}
})

// Server
app.listen(PORT, () => console.log(`Server running on port ${PORT}`))

```

classes.js

```
const fs = require('fs')

/**
* Class to manage reading and writing JSON files.
*/
class JSONFileManager {
/**
 * @param {string} filePath - The path to the JSON file.
 */
constructor(filePath) {
 this.filePath = filePath
}

/**
 * Reads the JSON file and returns its content.
 * @returns {Array|Object} The content of the JSON file.
 */
read() {
 if (fs.existsSync(this.filePath)) {
  const data = fs.readFileSync(this.filePath, 'utf-8')
  return JSON.parse(data || '[]')
 }
 return []
}

/**
 * Writes data to the JSON file.
 * @param {Array|Object} data - The data to write to the file.
 */
write(data) {
 fs.writeFileSync(this.filePath, JSON.stringify(data, null, 2), 'utf-8')
}
}

/**
* Class to manage menu items using a JSON file.
*/
class MenuManager {
/**
 * @param {string} filePath - The path to the JSON file.
 */
constructor(filePath) {
 this.fileManager = new JSONFileManager(filePath)
}

/**
 * Retrieves all menu items.
 * @returns {Array} The list of menu items.
 */
getAllMenuItems() {
```

```
 return this.fileManager.read()
}

getMenuItemById(menuItemId) {
 const menu = this.fileManager.read()
 return menu.find((m) => m.id === menuItemId)
}

/**
 * Adds a new menu item.
 * @param {Object} newMenuItem - The new menu item to add.
 * @returns {Object} The added menu item with an assigned ID.
 */
addMenuItem(newMenuItem) {
 const menu = this.fileManager.read()
 newMenuItem.id = `menu-item-${Date.now()}`
 menu.push(newMenuItem)
 this.fileManager.write(menu)
 return newMenuItem
}
 }

 /**
* Class to manage orders using a JSON file.
*/
 class OrderManager {
/**
 * @param {string} filePath - The path to the JSON file.
 */
constructor(filePath) {
 this.fileManager = new JSONFileManager(filePath)
}

/**
 * Retrieves all orders.
 * @returns {Array} The list of orders.
 */
getAllOrders() {
 return this.fileManager.read()
}

/**
 * Retrieves an order by ID.
 * @param {string} orderId - The ID of the order to retrieve.
 * @returns {Object|null} The order, or null if not found.
 */
getOrderById(orderId) {
 const orders = this.fileManager.read()
 return orders.find((o) => o.id === orderId)
}

/**
 * Adds a new order.
```

```
 * @param {Object} newOrder - The new order to add.
 * @returns {Object} The added order with an assigned ID.
 */
addOrder(newOrder) {
 const orders = this.fileManager.read()
 newOrder.id = `order-${Date.now()}`
 orders.push(newOrder)
 this.fileManager.write(orders)
 return newOrder
}

/**
 * Updates the status of an order.
 * @param {string} orderId - The ID of the order to update.
 * @param {string} status - The new status of the order.
 * @returns {Object|null} The updated order, or null if not found.
 */
updateOrderStatus(orderId, status) {
 const orders = this.fileManager.read()
 const order = orders.find((o) => o.id === orderId)
 if (!order) return null

 order.status = status
 this.fileManager.write(orders)
 return order
}

/**
 * Deletes an order.
 * @param {string} orderId - The ID of the order to delete.
 * @returns {Object|null} The deleted order, or null if not found.
 */
deleteOrder(orderId) {
 const orders = this.fileManager.read()
 const orderIndex = orders.findIndex((o) => o.id === orderId)
 if (orderIndex === -1) return null

 const deletedOrder = orders.splice(orderIndex, 1)[0]
 this.fileManager.write(orders)
 return deletedOrder
}
 }

 module.exports = { MenuManager, OrderManager }

```

Приклад одного предмету з меню:

```
{
 "name": "Cheeseburger",
 "image": "https://i.imgur.com/kDpWYVB.jpeg",
 "description": "A juicy cheeseburger with lettuce and tomato.",
 "price": 8.99,
```

```
  "category": "Burgers",
  "id": "menu-item-1733137777873"
}
```

Приклад одного замовлення:

```
{
 "items": [
  {
   "id": "menu-item-1733137777873",
   "quantity": 1
  },
  {
   "id": "menu-item-1733137777875",
   "quantity": 3
  }
 ],
 "customerDetails": {
  "name": "John Doe",
  "address": "123 Main St",
  "phone": "+3800442384470"
 },
 "totalPrice": 29.96,
 "status": "inactive",
 "id": "order-1733139019921"
}
```

Front-end

Весь вебсайт

layout.tsx:

```
import clsx from 'clsx'
import type { Metadata } from 'next'
import localFont from 'next/font/local'

import Providers from '@/app/providers'

import '../styles/globals.css'

const geistSans = localFont({
src: '../styles/fonts/GeistVF.woff',
variable: '--font-geist-sans',
weight: '100 900'
})
const geistMono = localFont({
src: '../styles/fonts/GeistMonoVF.woff',
variable: '--font-geist-mono',
weight: '100 900'
})

const metadata: Metadata = {
title: 'Restaurant Menu',
description: 'A simple restaurant menu app for IPZ lab9'
}
```

```tsx

 const RootLayout = ({
children
 }: Readonly<{
children: React.ReactNode
 }>) => {
return (
 <html lang="en">
  <body
   className={clsx(
    geistSans.variable,
    geistMono.variable,
    'm-0 h-full antialiased'
   )}
  >
   <Providers>{children}</Providers>
  </body>
 </html>
)
 }

 export { metadata }
 export default RootLayout
```

loading.tsx:

```tsx
 import LoadingComp from '@/components/loading/LoadingComp'

 const Loading = () => {
return <LoadingComp text="Loading..." />
 }

 export default Loading
```

providers.tsx

```tsx
 'use client'

 import { ReactNode, useState } from 'react'
 import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
 import { ReactQueryDevtools } from '@tanstack/react-query-devtools'
 import { ThemeProvider as NextThemeProvider } from 'next-themes'

const Providers = ({ children }: { children: ReactNode }) => {
const [queryClient] = useState(
 () =>
  new QueryClient({
   defaultOptions: {
    queries: {
     refetchOnWindowFocus: false,
     retryDelay: 1000 * 3, // 3 second
     staleTime: 1000 * 60 * 5 // 5 minutes
    }
```

```
    }
   })
 )

 return (
  <NextThemeProvider attribute="class">
   <QueryClientProvider client={queryClient}>
    {children}
    <ReactQueryDevtools initialIsOpen={false} />
   </QueryClientProvider>
  </NextThemeProvider>
 )
 }

 export default Providers

```

Сторінка з меню

layout.tsx

```
import { ReactNode } from 'react'

import MenuBar from '@/components/menu-bar/MenuBar'

const MenuLayout = ({
children
}: Readonly<{
children: ReactNode
}>) => {
 return (
  <>
   {children}
   <MenuBar />
  </>
 )
 }

 export default MenuLayout

```

page.tsx

```
 import CardSection from '@/app/(main)/components/CardSection'

 const Home = async () => {
return (
```

```
  <div className="flex flex-col items-center dark:bg-black">
   <h1 className="my-12 text-4xl font-extrabold dark:text-white">Menu</h1>
   <CardSection />
  </div>
 )
 }

 export default Home

```

CardSection.tsx

```
 'use client'

 import { useQuery } from '@tanstack/react-query'
 import Image from 'next/image'

 import ErrorComp from '@/components/error-comp/ErrorComp'
 import LoadingComp from '@/components/loading/LoadingComp'
 import { Button } from '@/components/ui/button'
 import {
Card,
CardContent,
CardDescription,
CardFooter,
CardHeader,
CardTitle
 } from '@/components/ui/card'
 import useCart from '@/hooks/use-cart'
 import { getMenu } from '@/services/api'
 import { MenuItem } from '@/types/menu'
 import addToCart from '@/utils/add-to-cart'

 const CardSection = () => {
const { cart, setCart } = useCart()
const fallback: MenuItem[] = []

const {
 data = fallback,
 error,
 isError,
 isLoading
} = useQuery({
 queryKey: ['menu'],
 queryFn: getMenu
})

if (isLoading) {
 return <LoadingComp text="Loading menu items..." />
}

if (isError) {
 return <ErrorComp message={error.message} keyArray={['menu']} />
```

```
}

return (
 <ul>
  {data.map(({ id, image, name, price, description }: MenuItem) => (
   <Card
    key={id}
    className="my-4 dark:border-neutral-600 dark:bg-neutral-900"
   >
    <CardHeader>
     <Image
      src={image.toString()}
      alt={name}
      width={300}
      height={200}
      className="rounded-lg"
     />
    </CardHeader>

    <CardContent>
     <CardTitle className="dark:text-white">{name}</CardTitle>
     <CardDescription className="max-w-72 py-1 text-amber-600">
      ${price}
     </CardDescription>
     <CardDescription className="max-w-72 py-1 dark:text-neutral-300">
      {description}
     </CardDescription>
    </CardContent>

    <CardFooter className="flex gap-x-2">
     <Button
      onClick={() => addToCart({ cart, itemId: id, setCart })}
      className="w-full"
     >
      Add to cart
     </Button>
    </CardFooter>
   </Card>
  ))}
 </ul>
)
 }

 export default CardSection

```

Сторінка з кошиком

layout.tsx

```tsx
import { ReactNode } from 'react'

const metadata = {
title: 'Cart'
}

const CartLayout = ({
children
}: Readonly<{
children: ReactNode
}>) => {
return (
<div className="flex flex-col items-center dark:bg-black">
<h1 className="my-12 text-4xl font-extrabold dark:text-white">Cart</h1>
{children}
</div>
)
}

export { metadata }
export default CartLayout

```

loading.tsx

```tsx
import LoadingComp from '@/components/loading/LoadingComp'

const Loading = () => {
return <LoadingComp text="Loading menu items..." />
}

export default Loading

```

page.tsx

```tsx
'use client'

import ErrorComp from '@/components/error-comp/ErrorComp'
import useCart from '@/hooks/use-cart'
import { CartItemProps } from '@/types/cart'

import CartFooterSection from './components/CartFooter'
import CartItem from './components/CartItem'
import EmptyCart from './components/EmptyCart'

const CartPage = () => {
const { cart, setCart, results } = useCart()

if (results.isLoading) {
return <p className="min-h-screen">Loading menu items...</p>
}
```

```

 if (results.isError) {
  return <ErrorComp message={results.error?.message} keyArray={['cart']} />
 }

 if (cart.length === 0) {
  return <EmptyCart />
 }

 const filteredResults = results.data.filter((dish) => dish !== undefined)

 return (
  <div className="mb-6 flex flex-col items-center gap-y-3 dark:bg-black">
   <ul className="mx-auto w-[90vw]">
    {cart.map((item: CartItemProps) => (
     <CartItem
      key={item.id}
      item={item}
      results={{ ...results, data: filteredResults }}
      cart={cart}
      setCart={setCart}
     />
    ))}
   </ul>

   <CartFooterSection
    cart={cart}
    results={{ ...results, data: filteredResults }}
   />
  </div>
 )
 }

 export default CartPage

```

EmptyCart.tsx

```
 const EmptyCart = () => {
return (
 <div className="flex min-h-screen flex-col items-center dark:bg-black">
  <p className="dark:text-white">Your cart is empty.</p>
 </div>
)
 }

 export default EmptyCart
```

CartItem.tsx

```tsx
import { Trash2 } from 'lucide-react'
import Image from 'next/image'

import { Button } from '@/components/ui/button'
import { Card, CardDescription, CardTitle } from '@/components/ui/card'
import { ActionProps, CartItemProps, CartItemResults } from '@/types/cart'
import removeItem from '@/utils/remove-from-cart'

interface CartItemFnProps {
item: CartItemProps
results: {
 data: CartItemResults[]
}
cart: ActionProps['cart']
setCart: ActionProps['setCart']
 }

 const CartItem = ({ item, results, cart, setCart }: CartItemFnProps) => {
const dishItem = results.data.find((dish) => dish?.id === item.id)

return (
 <Card
  key={item.id}
  className="my-3 flex p-3 dark:border-neutral-600 dark:bg-neutral-900"
 >
  <Image
   src={dishItem?.image?.toString() || '/default-image.png'}
   alt={dishItem?.name || 'Dish image'}
   width={100}
   height={100}
   className="rounded-md"
  />

  <div className="ml-2 flex flex-col justify-start">
   <CardTitle className="text-lg dark:text-white">
    {dishItem?.name}
   </CardTitle>
   <CardDescription>
    {item.quantity} item{item.quantity > 1 ? 's' : ''}
   </CardDescription>
   <CardDescription className="text-amber-600">
    ${(dishItem?.price ?? 0) * item.quantity}
   </CardDescription>
  </div>

  <Button
   variant="secondary"
   size="icon"
   className="ml-auto dark:bg-neutral-800"
   onClick={() =>
    removeItem({
     cart,
```

```
      setCart,
      itemId: item.id
     })
    }
   >
    <Trash2 strokeWidth={1.5} className="dark:text-white" />
   </Button>
  </Card>
 )
 }

 export default CartItem

```

CartFooter.tsx

```
 import Link from 'next/link'

 import { Button } from '@/components/ui/button'
 import { CartItemProps, CartItemResults } from '@/types/cart'

 const CartFooterSection = ({
cart,
results
 }: {
 cart: CartItemProps[]
 results: { data: CartItemResults[] }
 }) => {
 return (
  <>
   <div className="flex w-[80vw] justify-between">
    <p className="text-2xl dark:text-white">Total:</p>
    <b className="text-2xl text-amber-600">
     $
     {cart.reduce((acc, item) => {
      const dishItem = results.data.find((dish) => dish?.id === item.id)
      return acc + (dishItem?.price ?? 0) * item.quantity
     }, 0)}
    </b>
   </div>
   <Link href="/checkout" className="flex w-[80vw] justify-center">
    <Button>Checkout</Button>
   </Link>
  </>
 )
 }

 export default CartFooterSection

```

Сторінка з оформленням замовленням

layout.tsx

```tsx
import { ReactNode } from 'react'
import { ChevronLeft } from 'lucide-react'
import Link from 'next/link'

import { Button } from '@/components/ui/button'

const metadata = {
title: 'Checkout'
}

const CartLayout = ({
children
}: Readonly<{
children: ReactNode
}>) => {
return (
 <div className="flex flex-col items-center dark:bg-black">
  <Link href="/cart" className="absolute left-4 top-4">
   <Button variant="secondary" size="icon" className="rounded-full">
    <ChevronLeft strokeWidth={1.5} />
   </Button>
  </Link>
  <h1 className="my-12 text-4xl font-extrabold dark:text-white">
   Checkout
  </h1>
  {children}
 </div>
)
}

export { metadata }
export default CartLayout

```

page.tsx

```tsx
'use client'

import { useMutation } from '@tanstack/react-query'

import useCart from '@/hooks/use-cart'
import usePrepareOrder from '@/hooks/use-prepare-order'
import { sendOrder } from '@/services/api'
import { Customer } from '@/types/customer'

import OrderForm from './components/OrderForm'
import StatusAlerts from './components/StatusAlerts'
import TotalSection from './components/TotalSection'

const CheckoutPage = () => {
const { cart, setCart, results } = useCart()
```

```tsx
const prepareOrder = usePrepareOrder(cart, results)

const { isPending, mutate, status } = useMutation({
 mutationFn: async (order: Customer) => {
  const orderData = {
   ...prepareOrder(order),
   totalPrice: Number(prepareOrder(order).totalPrice),
   status: 'inactive' // or 'cooking' or 'completed' based on your logic
  }
  await sendOrder(orderData)
 },
 onError: (error) => {
  console.log('Error sending order', error)
 },
 onSuccess: () => {
  localStorage.removeItem('cart')
  setCart([])
 }
})

return (
 <div className="flex flex-col items-center gap-4">
  <TotalSection />
  <OrderForm isPending={isPending} mutate={mutate} />
  <StatusAlerts status={status} />
 </div>
)
 }

 export default CheckoutPage

```

TotalSection.tsx

```tsx
 import useCart from '@/hooks/use-cart'
 import { CartItemProps } from '@/types/cart'

 const TotalSection = () => {
const { cart, results } = useCart()

return (
 <div className="flex w-[70vw] justify-between">
  <p className="text-2xl">Total:</p>
  <b className="text-2xl text-amber-600">
   $
   {cart.reduce((acc: number, item: CartItemProps) => {
    const dishItem = results.data.find((dish) => dish?.id === item.id)
    return acc + (dishItem?.price ?? 0) * item.quantity
   }, 0)}
  </b>
 </div>
)
 }
```

```

export default TotalSection

```

OrderForm.tsx

```tsx
import { SubmitHandler, useForm } from 'react-hook-form'

import { Button } from '@/components/ui/button'
import { Input } from '@/components/ui/input'
import { Customer } from '@/types/customer'

const OrderForm = ({
isPending,
mutate
}: {
isPending: boolean
mutate: (data: Customer) => void
}) => {
const { register, handleSubmit } = useForm({
 defaultValues: {
  name: '',
  surname: '',
  address: '',
  phone: ''
 }
})

const handleCartSubmit: SubmitHandler<Customer> = (data) => mutate(data)

return (
 <form
  onSubmit={handleSubmit(handleCartSubmit)}
  className="flex w-[80vw] flex-col gap-4"
 >
  <Input type="text" placeholder="Your Name" {...register('name')} />
  <Input type="text" placeholder="Your Surname" {...register('surname')} />
  <Input type="text" placeholder="Your Address" {...register('address')} />
  <Input type="number" placeholder="Your Phone" {...register('phone')} />
  <Button disabled={isPending} type="submit">
   Submit
  </Button>
 </form>
)
}

export default OrderForm
```

StatusAlert.tsx

```tsx
import { CircleCheck } from 'lucide-react'

import { Alert, AlertDescription, AlertTitle } from '@/components/ui/alert'
```

```tsx

 const StatusAlerts = ({ status }: { status: string }) => {
return (
 <>
  {status === 'success' && (
   <Alert className="border border-green-500">
    <CircleCheck className="h-4 w-4" strokeWidth={1.5} />
    <AlertTitle>Your order has been sent!</AlertTitle>
    <AlertDescription>Thank you for your purchase!</AlertDescription>
   </Alert>
  )}
 </>
)
 }

 export default StatusAlerts

```

Сторінка для персоналу

layout.tsx

```tsx
import { ReactNode } from 'react'

import Navbar from './components/navbar/Navbar'

const metadata = {
title: 'Admin Page'
}

const CartLayout = ({
children
}: Readonly<{
children: ReactNode
}>) => {
return (
 <>
  <Navbar />
  {children}
 </>
)
}

export { metadata }
export default CartLayout

```

page.tsx

```tsx
'use client'

import { useQuery } from '@tanstack/react-query'

import ErrorComp from '@/components/error-comp/ErrorComp'
import LoadingComp from '@/components/loading/LoadingComp'
import { getOrders } from '@/services/api'
import { OrderProps } from '@/types/order'

import CardCompAdmin from './components/card/CardCompAdmin'

const AdminPage = () => {
const {
 data: orders,
 isLoading,
 isError,
 error
} = useQuery<OrderProps[]>({
 queryKey: ['orders'],
 queryFn: getOrders,
 staleTime: 1000 * 5
})

if (isLoading) {
```

```
  return <LoadingComp text="Loading orders..." />
 }

 if (isError) {
  return <ErrorComp message={error.message} keyArray={['orders']} />
 }

 return (
  <div className="grid grid-cols-1 items-baseline gap-4 p-3 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 2xl:grid-cols-5">
   {orders &&
    orders.map((order) => <CardCompAdmin key={order.id} order={order} />)}
  </div>
 )
 }

 export default AdminPage

```

Navbar.tsx

```
 import ReloadButton from './components/ReloadButton'

 const Navbar = () => {
 return (
  <nav className="flex items-center justify-between border-b bg-neutral-100 p-3 px-5 dark:border-neutral-700 dark:bg-neutral-800">
   <h1 className="text-xl font-bold">Admin</h1>
   <ReloadButton />
  </nav>
 )
 }

 export default Navbar

```

ReloadButton.tsx

```
 'use client'

 import { useQueryClient } from '@tanstack/react-query'

 import { Button } from '@/components/ui/button'

 const ReloadButton = () => {
 const queryClient = useQueryClient()

 return (
  <Button
   onClick={() =>
    queryClient.invalidateQueries({
     predicate: (query) => query.queryKey[0] === 'orders'
    })
   }
```

```
   disabled={queryClient.isFetching() > 0}
   variant="secondary"
   className="border border-neutral-600 dark:bg-neutral-700 dark:text-neutral-100"
  >
   Reload
  </Button>
 )
 }

 export default ReloadButton

```

CardCompAdmin.tsx

```
 import { useEffect, useState } from 'react'
 import { useQueryClient } from '@tanstack/react-query'
 import clsx from 'clsx'
 import { X } from 'lucide-react'

 import { Button } from '@/components/ui/button'
 import {
Card,
CardDescription,
 CardFooter,
 CardHeader,
 CardTitle
 } from '@/components/ui/card'
 import { deleteOrder } from '@/services/api'
 import { OrderProps } from '@/types/order'

 import CardOrderItems from './components/CardOrderItems'
 import StatusTabs from './components/StatusTabs'

 interface CardCompAdminProps {
order: OrderProps
key: string
 }

 const CardCompAdmin = ({ order }: CardCompAdminProps) => {
const queryClient = useQueryClient()
const [status, setStatus] = useState(order.status)

useEffect(() => setStatus(order.status), [order.status])

const handleDeleteOrder = () => {
 deleteOrder(order.id)
 queryClient.invalidateQueries({
  predicate: (query) => query.queryKey[0] === 'orders'
 })
}

return (
 <Card key={order.id}>
```

```
  <CardHeader>
   <div className="flex w-full justify-between">
    <CardTitle className="text-xl">{order.id.split('-')[1]}</CardTitle>
    <Button
     size="icon"
     variant="secondary"
     className={clsx(
      'right-2 top-2 rounded-full',
      status !== 'completed' && 'hidden'
     )}
     onClick={handleDeleteOrder}
    >
     <X strokeWidth={1.5} />
    </Button>
   </div>
   <CardDescription>${order.totalPrice}</CardDescription>
   <CardDescription>{order.customerDetails.phone}</CardDescription>
   <CardDescription>{order.customerDetails.name}</CardDescription>
   <CardDescription>{order.customerDetails.address}</CardDescription>
  </CardHeader>

  <hr className="border-t border-neutral-800" />

  <CardOrderItems order={order} />

  <CardFooter className="w-full justify-center">
   <StatusTabs status={status} id={order.id} />
  </CardFooter>
 </Card>
)
}

export default CardCompAdmin

```

CardOrderItems.tsx

```
import { useQueries } from '@tanstack/react-query'

import ErrorComp from '@/components/error-comp/ErrorComp'
import { CardContent, CardDescription } from '@/components/ui/card'
import { getMenuItem } from '@/services/api'
import { OrderProps } from '@/types/order'

const CardOrderItems = ({ order }: { order: OrderProps }) => {
const results = useQueries({
 queries: order.items.map((item) => ({
  queryKey: ['menu', item.id],
  queryFn: () => getMenuItem(item.id)
 })),
 combine: (data) => {
  return {
   data: data.map((result) => result.data),
```

```
    isLoading: data.some((result) => result.isLoading),
    isError: data.some((result) => result.isError),
    error: data.find((result) => result.isError)?.error
   }
  }
})

if (results.isLoading) {
 return <p>Loading...</p>
}

if (results.isError) {
 return <ErrorComp message={results.error!.message} keyArray={['menu']} />
}

return (
 <CardContent className="pt-4">
  {results.data.map((item, index) => (
   <CardDescription key={index} className="flex justify-between">
    <p>{item!.name}</p>
    <p>{order.items[index].quantity}</p>
   </CardDescription>
  ))}
 </CardContent>
)
}

export default CardOrderItems

```

StatusTabs.tsx

```
import { FC } from 'react'
import { useQueryClient } from '@tanstack/react-query'
import { CircleCheck, CircleSlash, CookingPot } from 'lucide-react'

import { Tabs, TabsList, TabsTrigger } from '@/components/ui/tabs'
import { updateOrderStatus } from '@/services/api'
import { OrderProps } from '@/types/order'

interface StatusTabsProps {
status: OrderProps['status']
id: string
}

const StatusTabs: FC<StatusTabsProps> = ({ status, id }) => {
const queryClient = useQueryClient()

return (
 <Tabs
  defaultValue={status}
  onValueChange={(value) => {
   updateOrderStatus(id, value)
```

```
    queryClient.invalidateQueries({
     predicate: (query) => query.queryKey[0] === 'orders'
    })
   }}
  >
   <TabsList className="flex justify-center">
    <TabsTrigger value="inactive">
     <CircleSlash strokeWidth={1.5} />
    </TabsTrigger>
    <TabsTrigger value="cooking">
     <CookingPot strokeWidth={1.5} />
    </TabsTrigger>
    <TabsTrigger value="completed">
     <CircleCheck strokeWidth={1.5} />
    </TabsTrigger>
   </TabsList>
  </Tabs>
 )
 }

 export default StatusTabs

```

Міжсторінкові компоненти

ErrorComp.tsx

```tsx
import { FC } from 'react'
import { useQueryClient } from '@tanstack/react-query'

import { Button } from '@/components/ui/button'

interface ErrorCompProps {
message?: string
keyArray: string[]
}

const ErrorComp: FC<ErrorCompProps> = ({
message = 'Unknown error',
keyArray
}) => {
const queryClient = useQueryClient()

const handleRetry = () =>
 queryClient.invalidateQueries({
  queryKey: keyArray
 })

return (
 <div className="flex min-h-screen w-56 flex-col items-center gap-y-2">
  <p>Error: {message}</p>
  <Button className="w-full" onClick={handleRetry}>
   Retry
  </Button>
 </div>
)
}

export default ErrorComp

```

LoadingComp.tsx

```tsx
import React, { FC } from 'react'

interface LoadingCompProps {
text: string
}

const LoadingComp: FC<LoadingCompProps> = ({ text }) => {
return (
 <div
  className="flex min-h-screen flex-col items-center justify-center dark:bg-black"
  aria-live="polite"
 >
  <p className="text-2xl font-bold">{text}</p>
 </div>
)
```

```
}

export default LoadingComp

```

MenuBar.tsx

```
import React from 'react'
import clsx from 'clsx'
import { ShoppingCart, Utensils } from 'lucide-react'

import MenuItem from './components/MenuItem'

const MenuBar = () => {
return (
 <ul
  role="menubar"
  className={clsx(
   'sticky bottom-0 z-10 grid grid-cols-2 justify-items-center p-5 backdrop-blur-lg',
   'border-t border-neutral-300 dark:border-neutral-700',
   'bg-neutral-50/90 dark:bg-neutral-900/90'
  )}
 >
  <MenuItem title="Menu" href="/">
   <Utensils strokeWidth={1.5} />
  </MenuItem>

  <MenuItem title="Cart" href="/cart">
   <ShoppingCart strokeWidth={1.5} />
  </MenuItem>
 </ul>
)
}

export default MenuBar

```

MenuItem.tsx

```
import { ReactNode } from 'react'
import Link from 'next/link'

interface MenuItemProps {
href: string
children: ReactNode
title?: string
}

const MenuItem = ({ href, children, title }: MenuItemProps) => {
return (
 <Link
  href={href}
  className="flex items-center gap-x-2 text-neutral-600 dark:text-neutral-200"
  role="menuitem"
```

```
  >
   {children}
   {title && <span>{title}</span>}
  </Link>
 )
 }

 export default MenuItem

```

use-cart.tsx

```tsx
import { useEffect, useState } from 'react'
import { useQueries } from '@tanstack/react-query'

import { getMenuItem } from '@/services/api'
import { CartItemProps } from '@/types/cart'

const useCart = () => {
const [cart, setCart] = useState<CartItemProps[]>([])

useEffect(() => {
 const cartData = localStorage.getItem('cart')
 setCart(cartData ? JSON.parse(cartData) : [])
}, [])

const results = useQueries({
 queries: cart.map((item: { id: string }) => ({
  queryKey: ['menu', item.id],
  queryFn: () => getMenuItem(item.id)
 })),
 combine: (data) => {
  return {
   data: data.map((result) => result.data),
   isLoading: data.some((result) => result.isLoading),
   isError: data.some((result) => result.isError),
   error: data.find((result) => result.isError)?.error
  }
 }
})

return { cart, setCart, results }
}

export default useCart

```

use-prepare-order.tsx

```tsx
import { useMemo, useState } from 'react'

import { CartItemProps } from '@/types/cart'
import { Customer } from '@/types/customer'
import { MenuItem } from '@/types/menu'

const usePrepareOrder = (
cart: CartItemProps[],
results: { data: (MenuItem | undefined)[] }
) => {
const [date] = useState(new Date())

const prepareOrder = (order: Customer) => {
 return {
```

```
  id: `order-${date}`,
  items: cart.map((item) => ({ id: item.id, quantity: item.quantity })),
  customerDetails: {
   name: order.name,
   surname: order.surname,
   address: order.address,
   phone: order.phone
  },
  status: 'inactive',
  totalPrice: cart
   .reduce((acc: number, item: CartItemProps) => {
    const dishItem = results.data.find((dish) => dish?.id === item.id)
    return acc + (dishItem?.price ?? 0) * item.quantity
   }, 0)
   .toFixed(2)
 }
}

return useMemo(() => prepareOrder, [cart, results, date])
 }

 export default usePrepareOrder

```

API-функції

```
import axios from 'axios'

import { MenuItem } from '@/types/menu'
import { OrderProps } from '@/types/order'

const BASE_URL = 'http://localhost:5000'
const axiosInstance = axios.create({
baseURL: BASE_URL
})

const getMenu = async () => {
const response = await axiosInstance.get<MenuItem[]>('/menu')
return response.data
}

const getMenuItem = async (id: string) => {
const response = await axiosInstance.get<MenuItem>(`/menu/${id}`)
return response.data
}

const getOrders = async () => {
const response = await axiosInstance.get<OrderProps[]>('/orders')
return response.data
}

const sendOrder = async (order: OrderProps) => {
const response = await axiosInstance.post('/orders', order)
return response.data
}

const updateOrderStatus = async (id: string, status: string) => {
const response = await axiosInstance.patch(`/orders/${id}`, { status })
return response.data
}

const deleteOrder = async (id: string) => {
const response = await axiosInstance.delete(`/orders/${id}`)
return response.data
}

export {
deleteOrder,
getMenu,
getMenuItem,
getOrders,
sendOrder,
updateOrderStatus
}
```

Типи

cart.ts

```ts
import { Dispatch, SetStateAction } from 'react'

import { MenuItem } from '@/types/menu'

interface CartItemProps {
id: MenuItem['id']
quantity: number
 }

 interface CartItemResults {
id: string
price: number
image?: URL | string
name?: string
 }

 interface ActionProps {
cart: CartItemProps[]
itemId: string | `menu-item-${number}`
setCart: Dispatch<SetStateAction<CartItemProps[]>>
 }

 export type { ActionProps, CartItemProps, CartItemResults }

```

customer.ts

```ts
 interface Customer {
name: string
surname: string
phone: string
address: string
 }

 export type { Customer }

```

menu.ts

```ts
 interface MenuItem {
name: string
image: URL
description: string
price: number
category: string
id: `menu-item-${number}`
 }

 export type { MenuItem }

```

order.ts

```ts
import { CartItemProps } from '@/types/cart'

import { Customer } from './customer'

interface OrderProps {
items: CartItemProps[]
customerDetails: Customer
totalPrice: number
status: 'inactive' | 'cooking' | 'completed'
id: string
}

export type { OrderProps }

```

Утиліти

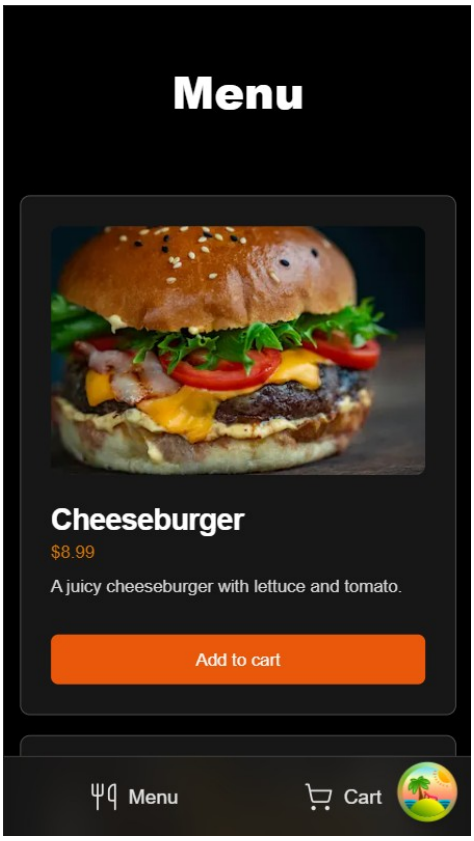
add-to-cart.ts

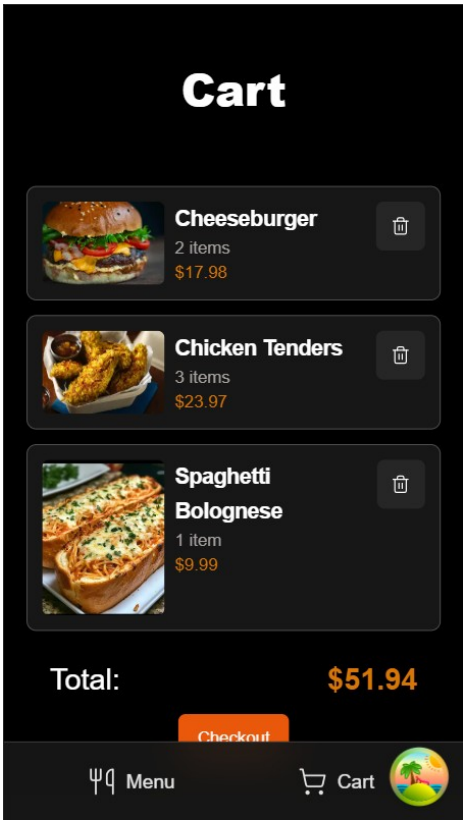
```ts
import { ActionProps } from '@/types/cart'

const addToCart = ({ cart, itemId, setCart }: ActionProps) => {
const updatedCart = [...cart]
const itemIndex = updatedCart.findIndex((item) => item.id === itemId)
console.log(itemId)

if (itemIndex !== -1) {
 updatedCart[itemIndex].quantity += 1
} else {
 updatedCart.push({ id: itemId, quantity: 1 })
}

localStorage.setItem('cart', JSON.stringify(updatedCart))
setCart(updatedCart)
 }

export default addToCart

```

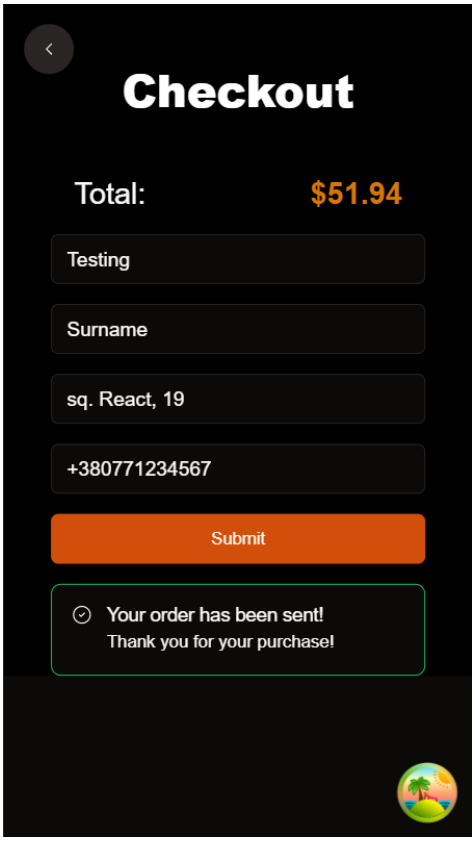
remove-from-cart.ts

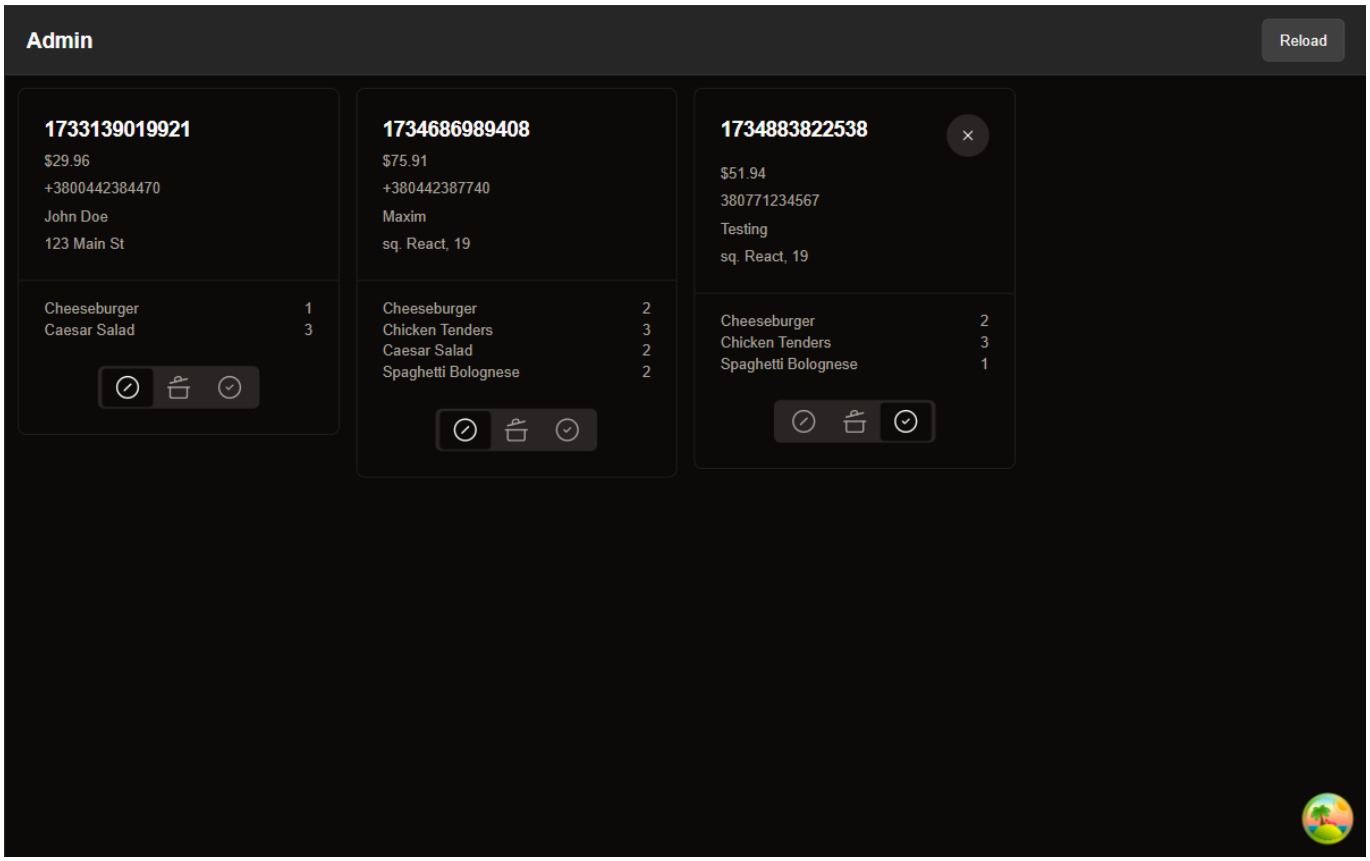
```ts
import { ActionProps } from '@/types/cart'

const removeItem = ({ cart, setCart, itemId }: ActionProps) => {
const newCart = cart.filter(({ id }) => id !== itemId)
localStorage.setItem('cart', JSON.stringify(newCart))
setCart(newCart)
 }

export default removeItem

```

Скриншоти:


Menu
Cheeseburger
$8.99
A juicy cheeseburger with lettuce and tomato.
Add to cart
Menu
Cart


Cart
Cheeseburger
2 items
$17.98
Chicken Tenders
3 items
$23.97
Spaghetti Bolognese
1 item
$9.99
Total:
$51.94
Checkout
Menu
Cart


Checkout
Total:
$51.94
Testing
Surname
sq. React, 19
+380771234567
Submit
Your order has been sent!
Thank you for your purchase!


Admin
Reload
1733139019921
$29.96
+3800442384470
John Doe
123 Main St
Cheeseburger 1
Caesar Salad 3
1734686989408
$75.91
+380442387740
Maxim
sq. React, 19
Cheeseburger 2
Chicken Tenders 3
Caesar Salad 2
Spaghetti Bolognese 2
1734883822538
$51.94
380771234567
Testing
sq. React, 19
Cheeseburger 2
Chicken Tenders 3
Spaghetti Bolognese 1

Висновки до лабораторної роботи:

1. **Вивчення архітектури "Клієнт-Сервер"**:
    - У ході виконання лабораторної роботи було розглянуто принципи роботи архітектурної моделі "Клієнт-Сервер".
    - Визначено переваги, такі як централізоване управління даними, розподіл функцій між клієнтом і сервером, що підвищує ефективність системи.

2. **Розробка системи керування замовленнями для ресторану**:
    - Реалізовано функціонал для клієнтів, включаючи перегляд меню, додавання товарів до кошика та оформлення замовлень.
    - Для персоналу реалізовано можливість перегляду замовлень, їх оновлення та видалення.

3. **Використання сучасних технологій**:
    - **Back-End**: використано Node.js та Express.js для створення REST API.
    - **Front-End**: застосовано бібліотеку React, серверний рендеринг Next.js, стилізація через TailwindCSS.
    - Реалізовано обробку форм за допомогою React Hook Form, управління станом через TanStack Query та локальне сховище.

4. **Дизайн та юзабіліті**:
    - Система має сучасний інтерфейс із динамічним оновленням даних, що покращує користувацький досвід.
    - Інтуїтивний розподіл функціоналу для клієнтів і персоналу.

5. **Проблеми та їх вирішення**:
    - Стикнулися з проблемами, пов'язаними з управлінням станом та синхронізацією даних між клієнтом і сервером. Ці проблеми були вирішені за допомогою TanStack Query та оптимізації запитів.
    - Проведено тестування API для виявлення та усунення помилок

6. **Практичні навички**:
    - Поглиблено знання у розробці веб-застосунків із використанням архітектури "Клієнт-Сервер".
    - Отримано досвід роботи з REST API, динамічним рендерингом компонентів і розробкою інтерактивного інтерфейсу.