

PROJECT

TOPIC

ເວັບສິອຕ້ວອນໄລນີ



START WITH WHY?



BUS TICKET WEB

เว็บซื้อตั๋วรถบัส

เราเชื่อว่าการเดินทางควรเป็นสิ่งที่สะดวกสบาย และ ไม่ควรเสียเวลาเดินทางไปยังสถานีรถบัสเพื่อซื้อตั๋ว

ด้วยเหตุนี้เราสร้างเว็บซื้อตั๋วรถบัส จ忙แก้ปัญหาเพื่อให้ผู้ใช้งานสามารถทำการซื้อตั๋วและเลือกที่นั่งได้อย่างสะดวกและรวดเร็ว อีกทั้งยังสามารถชำระเงินและอัพโหลดรูปสลิปการโอนเงินได้ในเว็บไซต์เดียว ทำให้เป็นการช่วยลดความยุ่งยากและเพิ่มความสะดวกสบายให้กับผู้ใช้งานที่ต้องการเดินทางด้วยรถบัส.

START WITH WHY?



BUS TICKET WEB

เว็บซื้อตั๋วรถบัส

อยากรู้การสร้างออกแบบ API การเลือกใช้ HTTP METHODS เช่น GET PUT POST DELETE ให้เหมาะสมกับงานต่างๆ เช่น Request แบบนี้จะ Response มาเป็นอะไร
จะมี Status Code แบบไหนสำหรับการซื้อตั๋วออนไลน์



BUS TICKET WEB

เป้าหมายหลักของเว็บซื้อตั๋วรถบัส

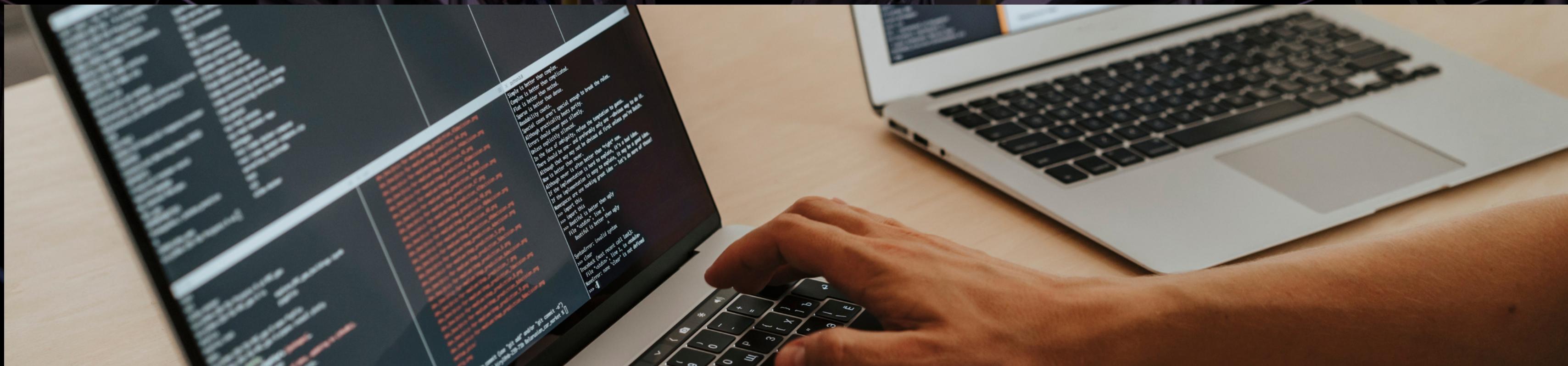


สะดวก รวดเร็ว ประหยัดเวลา และ สามารถเลือกซื้อตั๋วรถบัสได้อย่างง่ายดาย



BUS TICKET WEB

เป้าหมายหลักของการทำเว็บซื้อตั๋วรถบัส

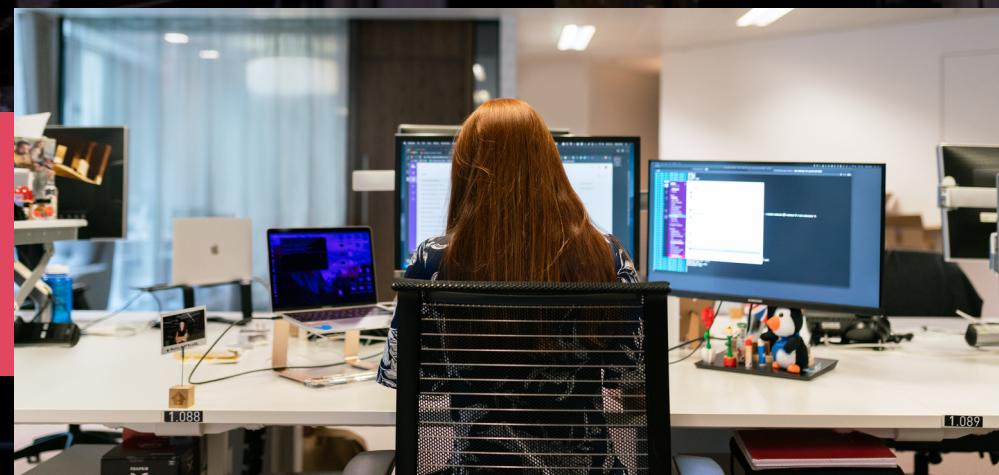
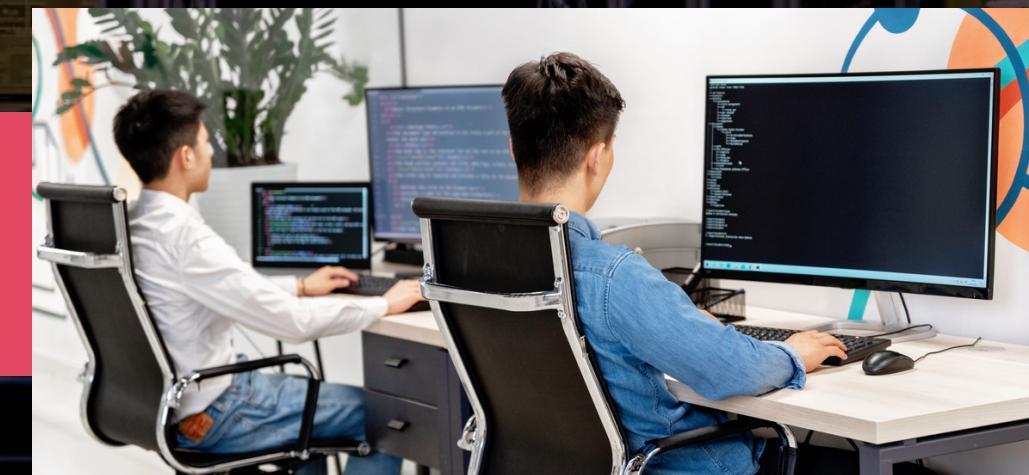


เพื่อให้สามารถเข้าใจในการสร้างหรือการใช้ API ในการซื้อตั๋วออนไลน์ได้ และสามารถออกแบบ API ให้เหมาะสมกับการใช้งานต่างๆ ได้อย่างมีประสิทธิภาพ



BUS TICKET WEB

ตัวอย่างการทำงาน



Front End

Front End ของเว็บซึ่งต้องรับส่วนที่ผู้ใช้งานสามารถเห็น และ เชื่อมต่อกับเว็บไซต์ได้โดยตรง ซึ่งประกอบไปด้วย PHP และ CSS ซึ่งมีหน้าที่รับข้อมูลจากผู้ใช้งานและแสดงผลข้อมูลให้กับผู้ใช้งานอย่างง่ายต่อการเข้าใจ

Back End

Backend ของเว็บซึ่งต้องรับส่งข้อมูลด้วย API ส่งข้อมูล HTTP และ Rust Web Actix Server ซึ่งเป็นตัวกลางในการติดต่อระหว่าง Frontend กับ Database

Database

DataBase ของเว็บซึ่งต้องรับส่งข้อมูล MySQL สำหรับเก็บข้อมูลโดยประกอบด้วย ข้อมูลของรถบัส ข้อมูลที่นั่ง ข้อมูลลูกค้า



BUS TICKET WEB

ROLE OUR TEAM



Front End & Database

Webserver
System Design & Presentation Design

Back End

API Design & Webserver
Main

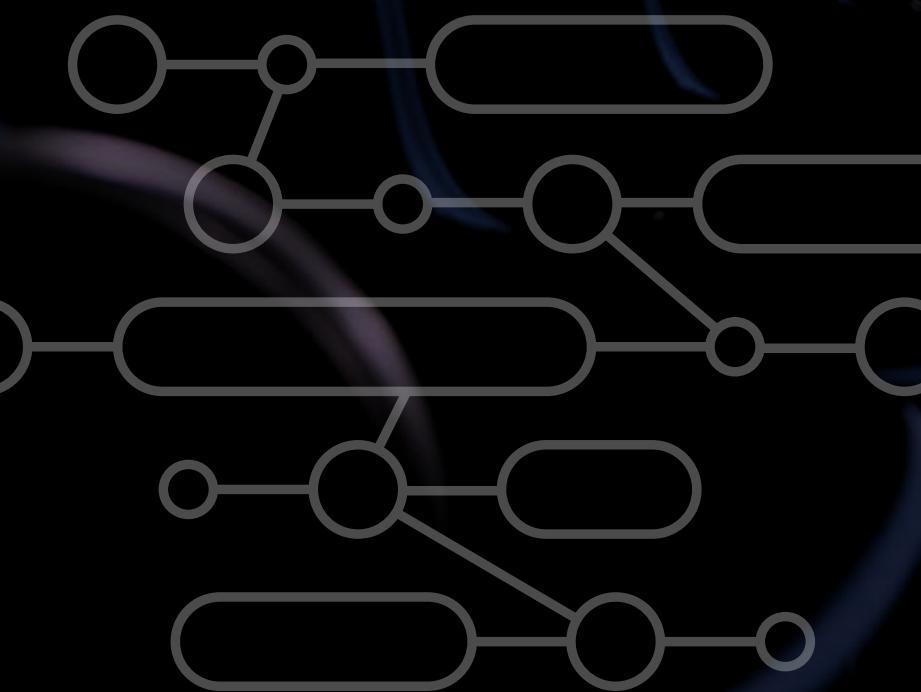
Over all & Presentation

API Design
Search info , Presentation



BUS TICKET WEB

DATABASE MySQL STRUCTURE





DATABASE MYSQL STRUCTURE

01 bus

เก็บข้อมูลรถบัส โดยมี

- id: ไอดีของรถบัส (primary_key)
- name: ชื่อของรถบัส
- direction: การเดินทาง
- outtime: เวลาออกรถ
- price: ราคาตั๋ว
- ticket: จำนวนตั๋ว

03 seat

เก็บข้อมูลที่นั่ง

- seatid: ไอดีของที่นั่ง (primary_key)
- seatnumber: หมายเลขที่นั่ง
- status: สถานะของที่นั่ง
- busid: ไอดีของรถบัส
- busname: ชื่อของรถบัส

02 customer

เก็บข้อมูลลูกค้า โดยมี

- id: ไอดีของลูกค้า (primary_key)
- name: ชื่อของลูกค้า
- contact: เบอร์โทรศัพท์ของลูกค้า
- date: วันเวลาที่ลูกค้าซื้อตั๋ว
- bus: ชื่อรถบัสที่ใช้เดินทาง
- derrection: เส้นทางการเดินทาง
- seatnumber: หมายเลขที่นั่ง

v	bus_ticket	customer
#	id : int(50)	
@	name : varchar(50)	
@	contact : varchar(50)	
@	date : varchar(50)	
@	bus : varchar(50)	
@	direction : varchar(50)	
#	seatnumber : int(20)	

v	bus_ticket	seat
#	seatid : int(20)	
#	seatnumber : int(20)	
#	status : int(11)	
#	busid : int(20)	
@	busname : varchar(50)	

v	bus_ticket	bus
#	id : int(20)	
@	name : varchar(50)	
@	direction : varchar(50)	
@	outtime : varchar(50)	
#	price : int(20)	
#	ticket : int(20)	



BUS TICKET WEB

RUST FILE STRUCTURE



RUST EXECUTABLE FILE STRUCTURE

main.rs เป็นไฟล์ที่เริ่มต้นการทำงานดของแอปพลิเคชัน โดยโหลดโมดูลต่าง ๆ

routes

01

bus_routes.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของรถบัส ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการเดินทาง, อัพเดตข้อมูลของรถบัส

02

customer_routes.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของลูกค้า ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการเดินทาง, เพิ่มข้อมูล และ ลบข้อมูลของลูกค้า

03

seat_routes.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของที่นั่งบนรถ ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการเดินทาง, อัพเดตข้อมูลของที่นั่งบนรถ

models

05

bus.rs

ไฟล์นี้เก็บໂນໂດລຂອງข้อมูลรถบัส ประกอบด้วย struct Bus ดังนี้

- id: ໄອດີຂອງรถบัส
- name: ชื่อของรถบัส
- direction: การเดินทาง
- outtime: เวลาออกรถ
- price: ราคาตั๋ว
- ticket: ຈຳນວນຕົ້ນ

06

customer.rs

ไฟล์นี้เก็บໂນໂດລຂອງข้อมูลลูกค้า ประกอบด้วย struct Customer ดังนี้

- id: ໄອດີຂອງลูกค้า
- name: ชื่อของลูกค้า
- contact: ແອຣໂກສະເພາະຂອງลูกค้า
- date: ວັນເວລາທີ່ລູກຄ້າເຂົ້າຫຼື້ວ
- bus: ຜ່ອມບັນດາທີ່ໃຊ້ເດີນການ
- seatnumber: ໄມຍເລກທີ່ນັ້ງ

07

seat.rs

ไฟล์นี้เก็บໂນໂດລຂອງข้อมูลທີ່ນັ້ງບ່ຽນຮັກ ประกอบด้วย struct Seat ดังนี้

- seatid: ໄອດີຂອງທີ່ນັ້ງ
- seatnumber: ໄມຍເລກທີ່ນັ້ງ
- status: ສະຖານະຂອງທີ່ນັ້ງ
- busid: ໄອດີຂອງรถบัส
- busname: ชื่อของรถบัส

handler

09

bus_handler.rs

สำหรับจัดการข้อมูลของ Bus ซึ่งประกอบด้วยຝັກສັນຕໍ່າງໆ เช่น get_bus() สำหรับດັ່ງຂໍ້ມູນຄັບສົ່ງ, update_bus_ticket() สำหรับອັບເດັກຂໍ້ມູນຈຳນວນຕົ້ນ

10

customer_handler.rs

สำหรับจัดการข้อมูลของ Customer ซึ่งประกอบด้วยຝັກສັນຕໍ່າງໆ เช่น get_customer_by_id() สำหรับແສດງຂໍ້ມູນລູກຄ້າຕາມໄອດີ, add_customer() สำหรับເພີ່ມຂໍ້ມູນລູກຄ້າ, delete_customer_by_id() สำหรับລົບຂໍ້ມູນລູກຄ້າອອກຈາກຮະບບຕາມໄອດີທີ່ກຳເຫັນ

11

seat_handler.rs

สำหรับจัดการข้อมูลของ Seat ซึ่งประกอบด้วยຝັກສັນຕໍ່າງໆ เช่น get_seat_by_id() สำหรับແສດງຂໍ້ມູນທີ່ນັ້ງຕາມໄອດີຂອງຮຽນ, update_seat_status() สำหรับອັບເດັກຂໍ້ມູນສະຖານະທີ່ນັ້ງ

mod.rs เป็นไฟล์ที่ใช้สำหรับการรวม import ของโมดูลในโพลเดอร์นี้ เพื่อให้ง่ายต่อการ import ในไฟล์อื่น ๆ ในโปรแกรม

04

routes/mod.rs

08

models/mod.rs

12

handler/mod.rs

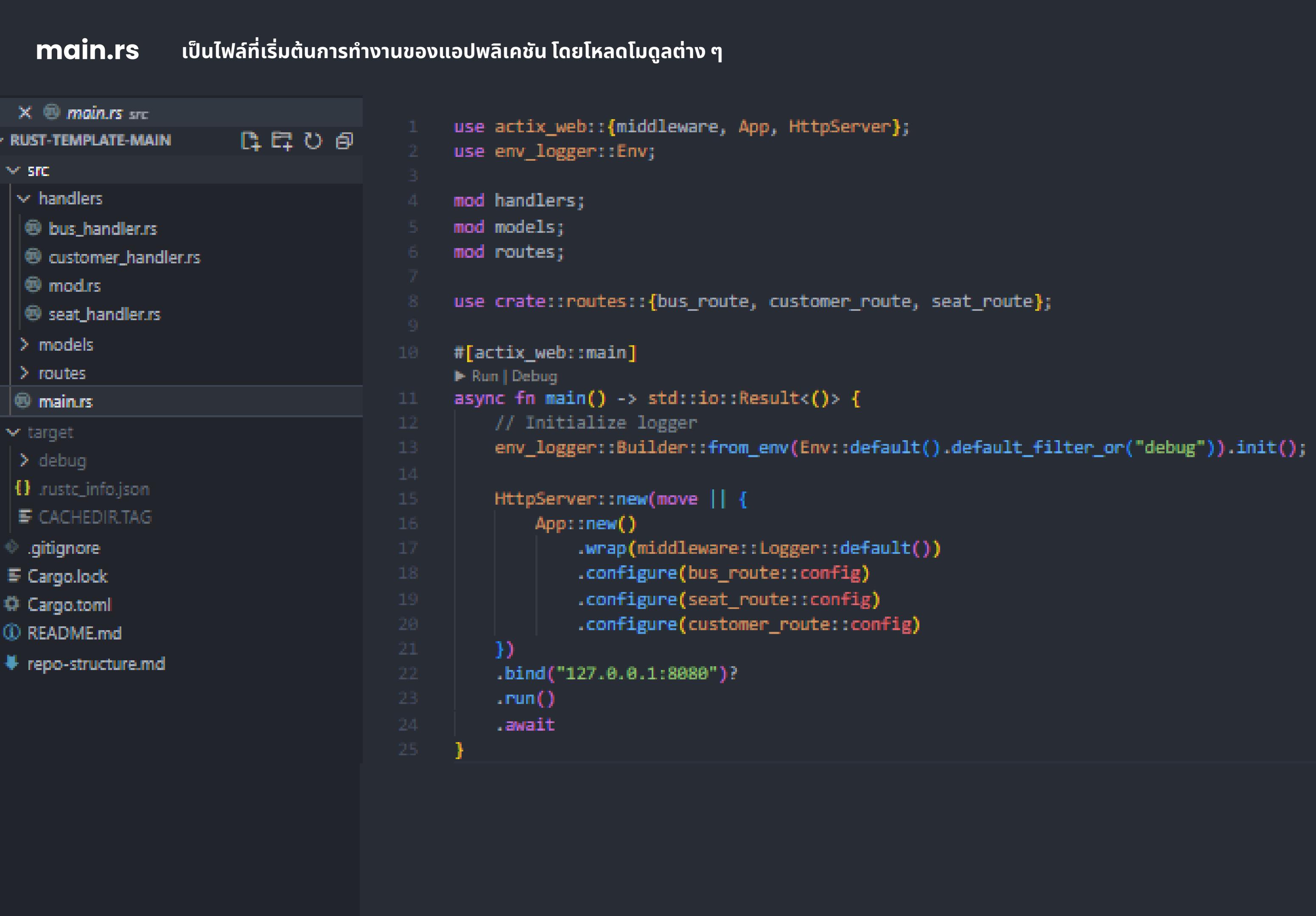


BUS TICKET WEB

MAIN.RS



main.rs เป็นไฟล์ที่เริ่มต้นการทำงานของแอปพลิเคชัน โดยโหลดโมดูลต่าง ๆ



```
main.rs src
RUST-TEMPLATE-MAIN
SRC
  handlers
    bus_handler.rs
    customer_handler.rs
    mod.rs
    seat_handler.rs
  models
  routes
main.rs
target
  debug
.rustc_info.json
CACHEDIR.TAG
.gitignore
Cargo.lock
Cargo.toml
README.md
repo-structure.md

1 use actix_web::{middleware, App, HttpServer};
2 use env_logger::Env;
3
4 mod handlers;
5 mod models;
6 mod routes;
7
8 use crate::routes::{bus_route, customer_route, seat_route};
9
10 #[actix_web::main]
11 ▶ Run | Debug
12 async fn main() -> std::io::Result<()> {
13     // Initialize logger
14     env_logger::Builder::from_env(Env::default().default_filter_or("debug")).init();
15
16     HttpServer::new(move || {
17         App::new()
18             .wrap(middleware::Logger::default())
19             .configure(bus_route::config)
20             .configure(seat_route::config)
21             .configure(customer_route::config)
22     })
23     .bind("127.0.0.1:8080")?
24     .run()
25     .await
}
```



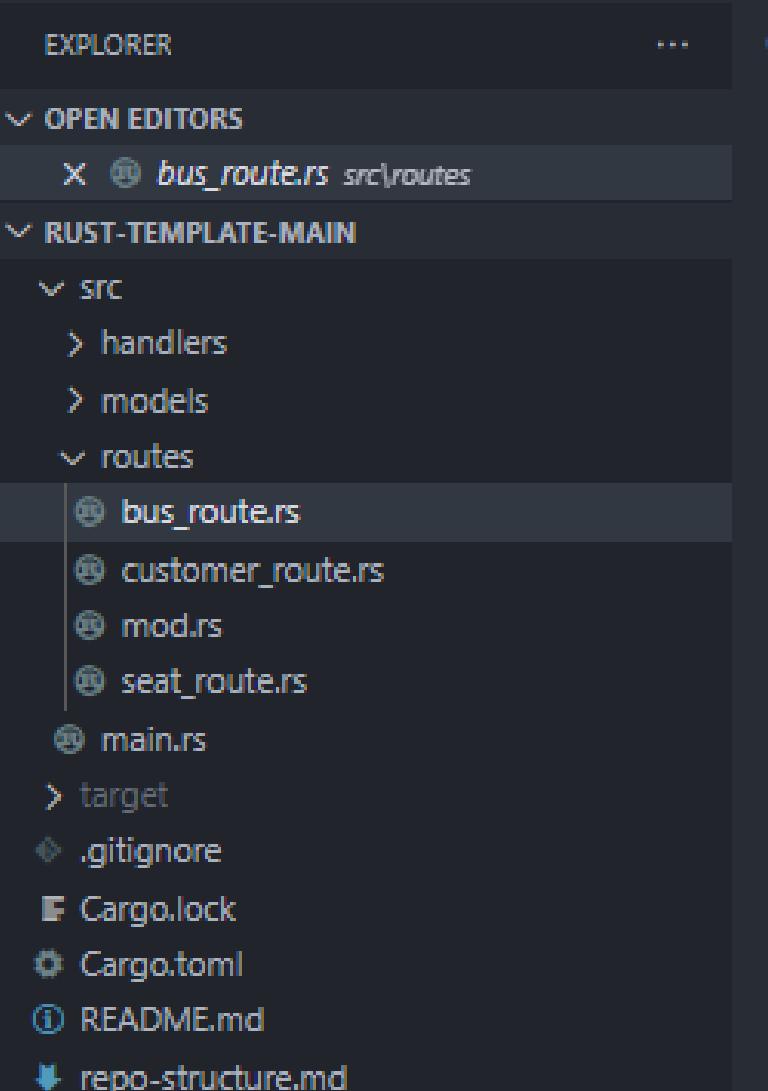
BUS TICKET WEB

ROUTES



routes/bus_routes.rs

เป็นไฟล์ที่เริ่มต้นการทำงานของแอปพลิเคชัน โดยโหลดโมดูลต่าง ๆ



routes

01

bus_route.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของรถบัส ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการดึงข้อมูล หรือ แก้ไขข้อมูลของรถบัส

02

customer_route.rs

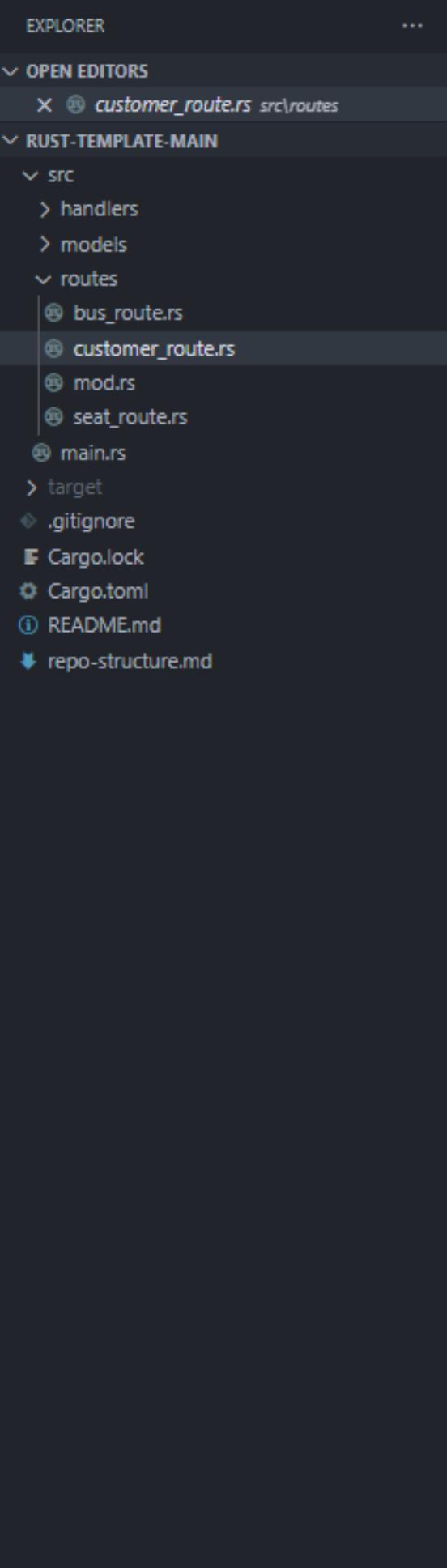
สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของลูกค้า ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการดึงข้อมูล, เพิ่มข้อมูล หรือ ลบข้อมูลของลูกค้า

03

seat_route.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของที่นั่งบนรถ ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการดึงข้อมูล หรือ แก้ไขข้อมูลของที่นั่ง

routes/customer_routes.rs เป็นไฟล์ที่เริ่มต้นการกำหนดการทำงานของแอปพลิเคชัน โดยโหลดโมดูลต่าง ๆ



routes

01

bus_route.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของรถบัส ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการดึงข้อมูล หรือ แก้ไขข้อมูลของรถบัส

02

customer_route.rs

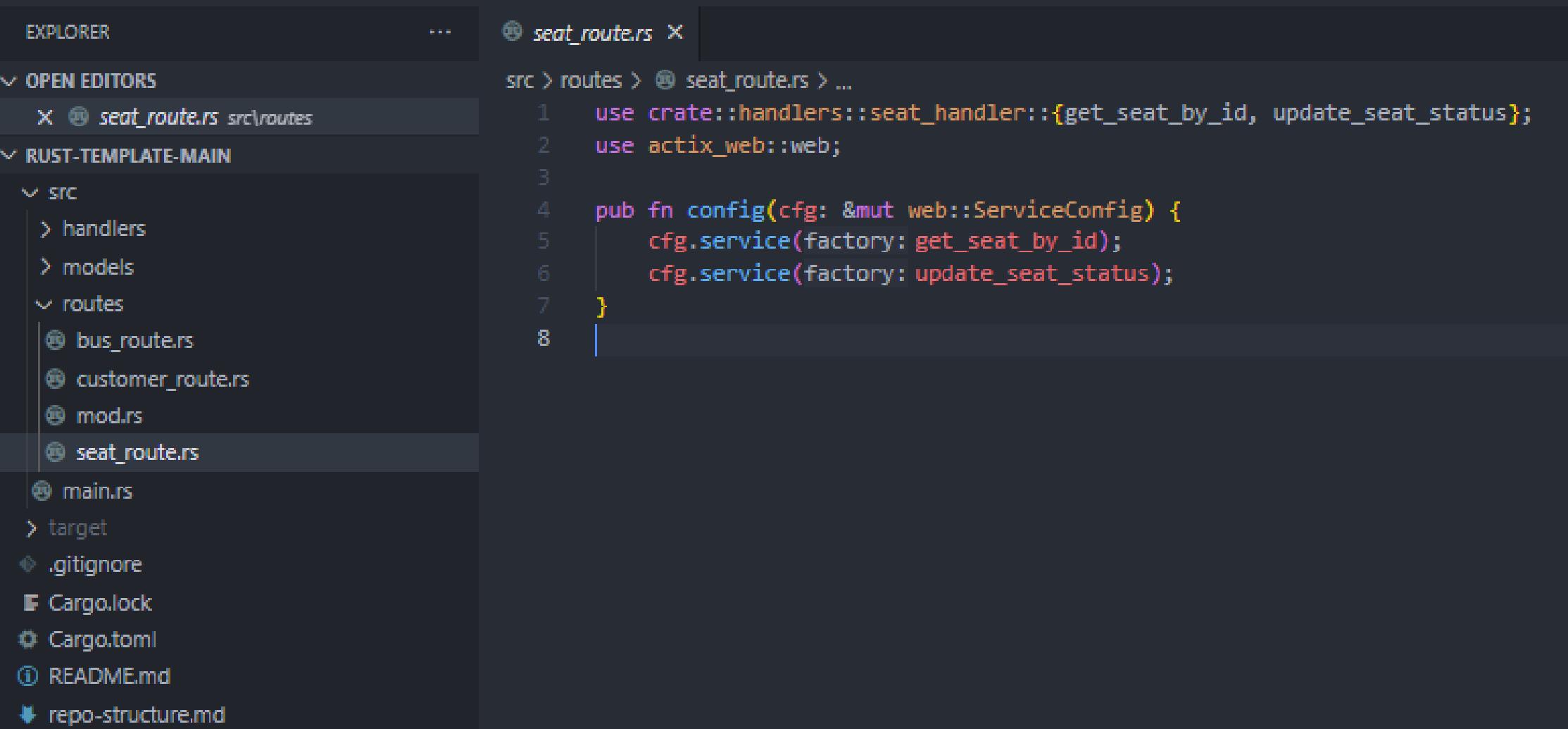
สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของลูกค้า ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการดึงข้อมูล, เพิ่มข้อมูล หรือ ลบข้อมูลของลูกค้า

03

seat_route.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของที่นั่งบนรถ ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการดึงข้อมูล หรือ แก้ไขข้อมูลของที่นั่ง

routes/seat_routes.rs เป็นไฟล์ที่เริ่มต้นการทำงานของแอปพลิเคชัน โดยโหลดโมดูลต่อๆ กัน



```
use crate::handlers::seat_handler::{get_seat_by_id, update_seat_status};
use actix_web::web;

pub fn config(cfg: &mut web::ServiceConfig) {
    cfg.service(factory: get_seat_by_id);
    cfg.service(factory: update_seat_status);
}
```

routes

01

bus_route.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของรถบัส ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการดึงข้อมูล หรือ แก้ไขข้อมูลของรถบัส

02

customer_route.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของลูกค้า ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการดึงข้อมูล, เพิ่มข้อมูล หรือ ลบข้อมูลของลูกค้า

03

seat_route.rs

สำหรับกำหนดเส้นทาง API สำหรับการจัดการข้อมูลของที่นั่งบนรถ ไฟล์นี้จะรับผิดชอบการกำหนดเส้นทาง API ที่เกี่ยวข้องกับการดึงข้อมูล หรือ แก้ไขข้อมูลของที่นั่ง



BUS TICKET WEB

MODELS



models/bus.rs

เป็นไฟล์ที่เริ่มต้นการทำงานของแอปพลิเคชัน โดยโหลดโมดูลต่าง ๆ

The screenshot shows a dark-themed code editor with the following file structure:

- EXPLORER
- OPEN EDITORS
- RUST-TEMPLATE-MAIN
- SRC
 - handlers
 - models
 - bus.rs
 - customer.rs
 - mod.rs
 - seat.rs
 - routes
 - main.rs
- target
- .gitignore
- Cargo.lock
- Cargo.toml
- README.md
- repo-structure.md

The content of the bus.rs file is:

```
use serde::{Deserialize, Serialize};  
#[derive(Serialize, Deserialize)]  
impl Bus {  
    pub struct Bus {  
        pub id: i32,  
        pub name: String,  
        pub direction: String,  
        pub outtime: String,  
        pub price: i32,  
        pub ticket: i32,  
    }  
}
```

models

05

bus.rs

ไฟล์นี้เก็บโมเดลของข้อมูลรถบัส ประกอบด้วย struct Bus ดังนี้

- id: ไอดีของรถบัส
- name: ชื่อของรถบัส
- direction: การเดินทาง
- outtime: เวลาออกรถ
- price: ราคาตั๋ว
- ticket: จำนวนตั๋ว

06

customer.rs

ไฟล์นี้เก็บโมเดลของข้อมูลลูกค้า ประกอบด้วย struct Customer ดังนี้

- id: ไอดีของลูกค้า
- name: ชื่อของลูกค้า
- contact: เบอร์โทรศัพท์ของลูกค้า
- date: วันเวลาที่ลูกค้าซื้อตั๋ว
- bus: ชื่อรถบัสที่ใช้เดินทาง
- seatnumber: หมายเลขที่นั่ง

07

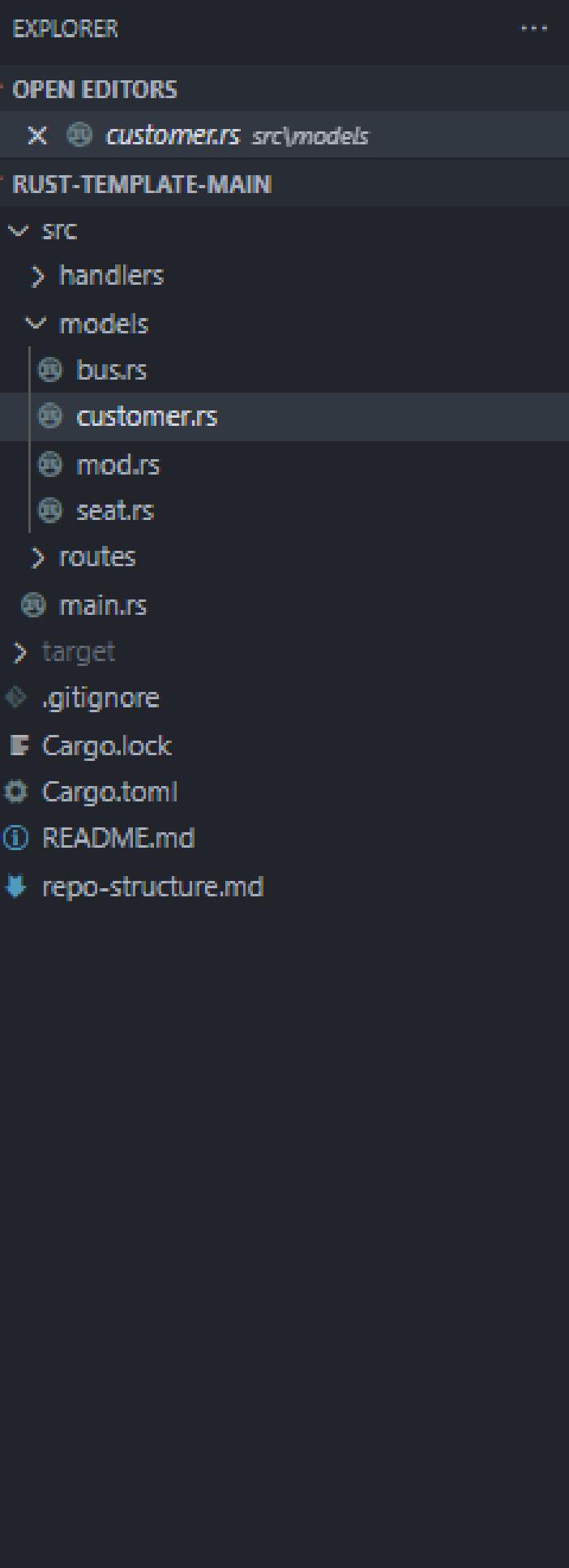
seat.rs

ไฟล์นี้เก็บโมเดลของข้อมูลที่นั่งบนรถ ประกอบด้วย struct Seat ดังนี้

- seatid: ไอดีของที่นั่ง
- seatnumber: หมายเลขที่นั่ง
- status: สถานะของที่นั่ง
- busid: ไอดีของรถบัส
- busname: ชื่อของรถบัส

models/customer.rs

เป็นไฟล์ที่เริ่มต้นการกำหนดงานของแอปพลิเคชัน โดยโหลดโมดูลต่าง ๆ



models

05

bus.rs

ไฟล์นี้เก็บโมเดลของข้อมูลรถบัส ประกอบด้วย struct Bus ดังนี้

- id: ไอดีของรถบัส
- name: ชื่อของรถบัส
- direction: การเดินทาง
- outtime: เวลาออกรถ
- price: ราคาตั๋ว
- ticket: จำนวนตั๋ว

06

customer.rs

ไฟล์นี้เก็บโมเดลของข้อมูลลูกค้า ประกอบด้วย struct Customer ดังนี้

- id: ไอดีของลูกค้า
- name: ชื่อของลูกค้า
- contact: เบอร์โทรศัพท์ของลูกค้า
- date: วันเวลาที่ลูกค้าซื้อตั๋ว
- bus: ชื่อรถบัสที่ใช้เดินทาง
- seatnumber: หมายเลขที่นั่ง

07

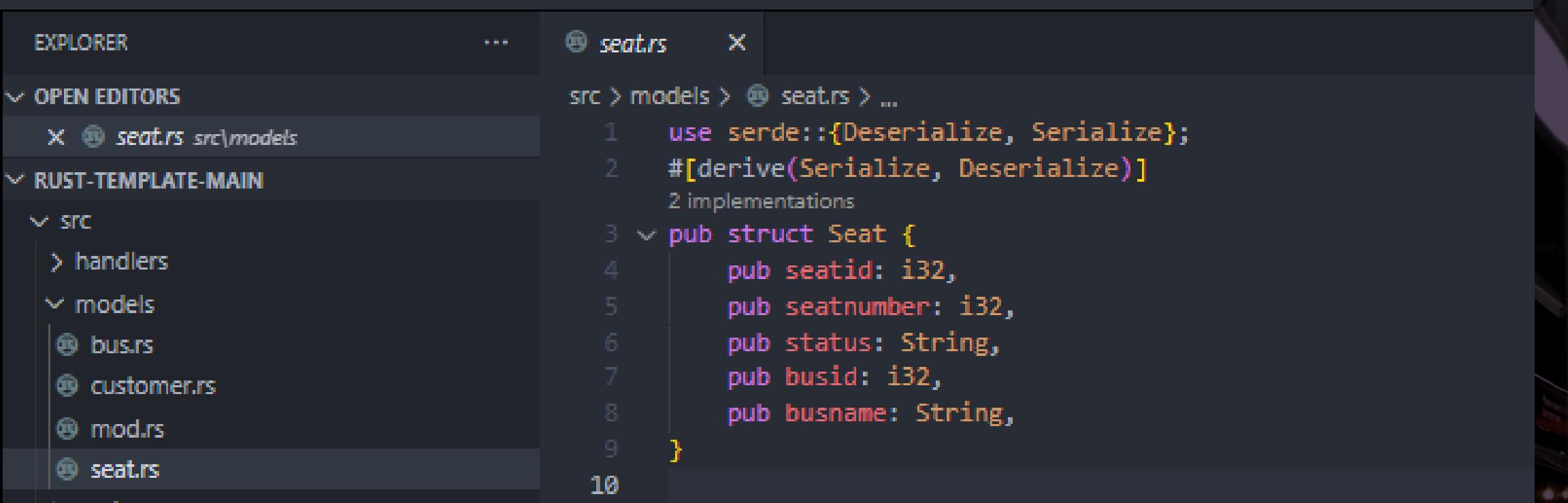
seat.rs

ไฟล์นี้เก็บโมเดลของข้อมูลที่นั่งบนรถ ประกอบด้วย struct Seat ดังนี้

- seatid: ไอดีของที่นั่ง
- seatnumber: หมายเลขที่นั่ง
- status: สถานะของที่นั่ง
- busid: ไอดีของรถบัส
- busname: ชื่อของรถบัส

models/seat.rs

เป็นไฟล์ที่เริ่มต้นการทำงานของแอปพลิเคชัน โดยโหลดโมดูลต่าง ๆ



```
use serde::{Deserialize, Serialize};
#[derive(Deserialize, Serialize)]
pub struct Seat {
    pub seatid: i32,
    pub seatnumber: i32,
    pub status: String,
    pub busid: i32,
    pub busname: String,
}
```

models

05

bus.rs

ไฟล์นี้เก็บโมเดลของข้อมูลรถบัส ประกอบด้วย struct Bus ดังนี้

- id: ไอดีของรถบัส
- name: ชื่อของรถบัส
- direction: การเดินทาง
- outtime: เวลาออกรถ
- price: ราคาตั๋ว
- ticket: จำนวนตั๋ว

customer.rs

ไฟล์นี้เก็บโมเดลของข้อมูลลูกค้า ประกอบด้วย struct Customer ดังนี้

- id: ไอดีของลูกค้า
- name: ชื่อของลูกค้า
- contact: เบอร์โทรศัพท์ของลูกค้า
- date: วันเวลาที่ลูกค้าซื้อตั๋ว
- bus: ชื่อรถบัสที่ใช้เดินทาง
- seatnumber: หมายเลขที่นั่ง

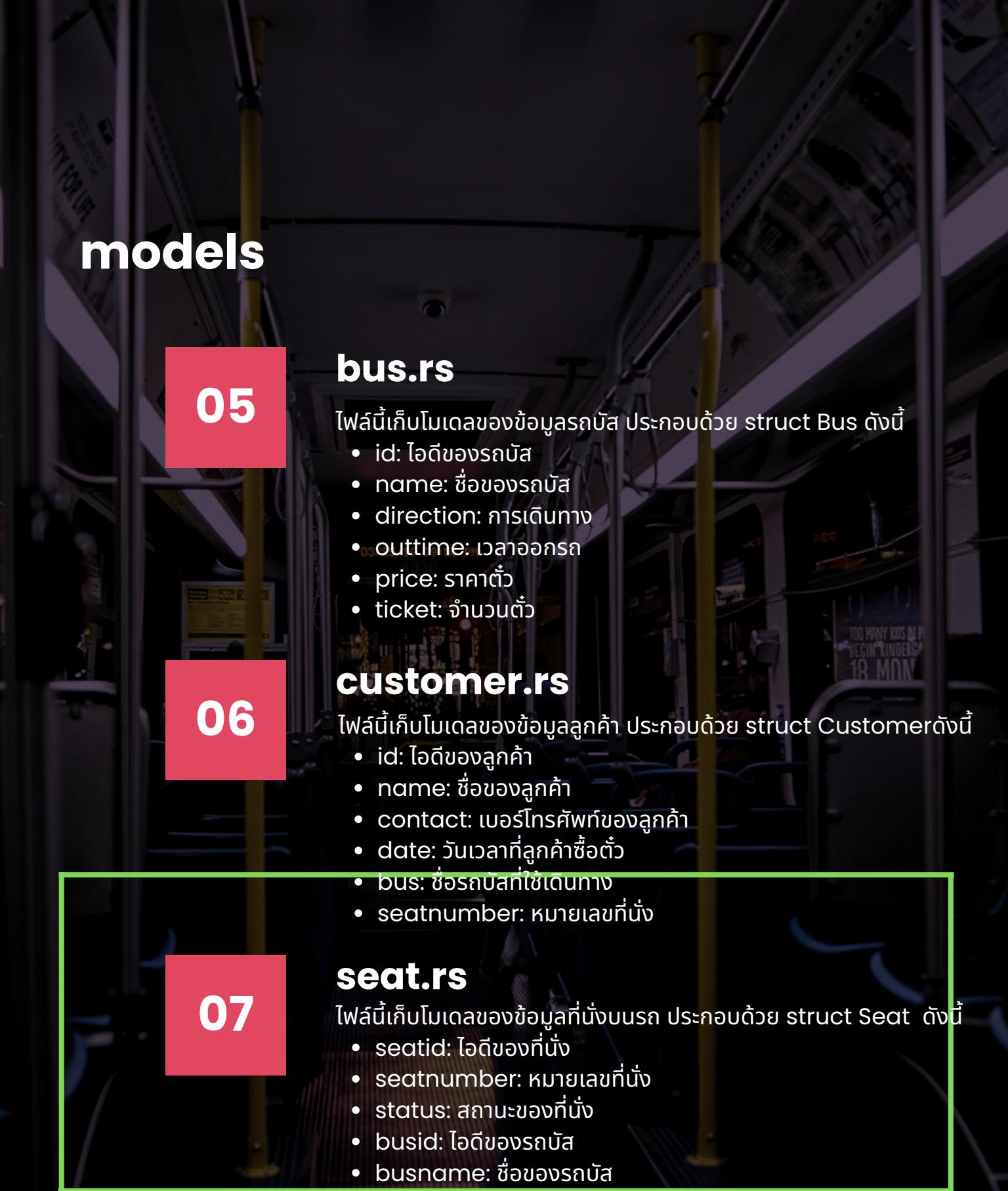
06

seat.rs

ไฟล์นี้เก็บโมเดลของข้อมูลที่นั่งบนรถ ประกอบด้วย struct Seat ดังนี้

- seatid: ไอดีของที่นั่ง
- seatnumber: หมายเลขที่นั่ง
- status: สถานะของที่นั่ง
- busid: ไอดีของรถบัส
- busname: ชื่อของรถบัส

07





BUS TICKET WEB

HANDLERS

handlers/bus_handler.rs จะเป็นไฟล์ที่ใช้สำหรับการจัดการกับข้อมูลต่างๆ ในระบบ

```
EXPLORER ... customer_handler.rs mod.rs .../models mod.rs .../routes bus_handler.rs X  
OPEN EDITORS  
customer_handler.rs src/handlers  
mod.rs src/models  
mod.rs src/routes  
X bus_handler.rs src/handlers  
RUST-TEMPLATE-MAIN D+ E O O  
src  
handlers  
bus_handler.rs  
customer_handler.rs  
mod.rs  
seat_handler.rs  
models  
routes  
bus_routes.rs  
customer_routes.rs  
mod.rs  
seat_routes.rs  
main.rs  
target  
.gitignore  
Cargo.lock  
Cargo.toml  
README.md  
repo-structure.md  
src > handlers > bus_handler.rs > update_bus_ticket  
6     async fn get_bus() -> impl Responder {  
7         let response_body: Vec<Bus> = vec![  
8             Bus {  
9                 id: 1,  
10                name: "Busbin".to_string(),  
11                direction: "พัฒนา-ครุฑ์".to_string(),  
12                outtime: "22:00".to_string(),  
13                price: 800,  
14                ticket: 20,  
15            },  
16            Bus {  
17                id: 2,  
18                name: "Buspong".to_string(),  
19                direction: "พัฒนา-สระบุรี".to_string(),  
20                outtime: "14:00".to_string(),  
21                price: 600,  
22                ticket: 20,  
23            },  
24        ];  
25        HttpResponse::Ok().json(response_body)  
26    }  
27  
#[put("/bus")]  
28     async fn update_bus_ticket(bus: web::Json<Bus>) -> impl Responder {  
29         let new_ticket: i32 = bus.ticket;  
30  
31         if bus.id == 1 {  
32             let response_body: Bus = Bus {  
33                 id: bus.id,  
34                 name: "Busbin".to_string(),  
35                 direction: "พัฒนา-ครุฑ์".to_string(),  
36                 outtime: "22:00".to_string(),  
37                 price: 800,  
38                 ticket: new_ticket,  
39             };  
40  
41             HttpResponse::Ok().json(response_body)  
42         } else {  
43             HttpResponse::NotFound().json(json!({  
44                 "message": "Bus ID invalid"  
45             }))  
46         }  
47     }  
48 }
```

handler

09

bus_handler.rs

สำหรับจัดการข้อมูลของ Bus ซึ่งประกอบด้วยฟังก์ชันต่างๆ เช่น get_bus() สำหรับดึงข้อมูลรถบัสกั้นๆ หนึ่งคัน, update_bus_ticket() สำหรับอัปเดตข้อมูลจำนวนตั๋ว

10

customer_handler.rs

สำหรับจัดการข้อมูลของ Customer ซึ่งประกอบด้วยฟังก์ชันต่างๆ เช่น get_customer_by_id() สำหรับแสดงข้อมูลลูกค้าตามไอดี, add_customer() สำหรับเพิ่มข้อมูลลูกค้า, delete_customer_by_id() สำหรับลบข้อมูลลูกค้าออกจากระบบตามไอดีที่กำหนด

11

seat_handler.rs

สำหรับจัดการข้อมูลของ Seat ซึ่งประกอบด้วยฟังก์ชันต่างๆ เช่น get_seat_by_id() สำหรับแสดงข้อมูลที่นั่งตามไอดีของรถบัส, update_seat_status() สำหรับอัปเดตข้อมูลสถานะที่นั่ง

handlers/customer_handler.rs จะเป็นไฟล์ที่ใช้สำหรับการจัดการกับข้อมูลต่างๆ ในระบบ

```
EXPLORER ...  
OPEN EDITORS ...  
customer_handler.rs x mod.rs ...\\models mod.rs ...\\routes  
src > handlers > customer_handler.rs > add_customer  
1 use crate::models::customer::Customer;  
2 use actix_web::{delete, get, post, web, HttpResponse, Responder};  
3 use serde_json::json;  
4  
5 #[get("/customer/{id}")]  
6 async fn get_customer_by_id(id: web::Path<i32>) -> impl Responder {  
7     let customerid: i32 = id.into_inner();  
8  
9     if customerid == 2 {  
10         let response_body: Customer = Customer {  
11             id: customerid,  
12             bus: "Busbin".to_string(),  
13             seatnumber: 6,  
14             name: "JonJack".to_string(),  
15             date: "2023-03-20".to_string(),  
16             contact: "0844758262".to_string(),  
17         };  
18         HttpResponse::Ok().json(response_body)  
19     } else {  
20         return HttpResponse::NotFound().json(json!{  
21             "message": "Customer not found"  
22         });  
23     }  
24 }  
25  
26 #[post("/customer")]  
27 async fn add_customer(customer: web::Json<Customer>) -> impl Responder {  
28     if customer.name.is_empty() || customer.contact.is_empty() {  
29         return HttpResponse::BadRequest().json(json!{  
30             "message": "Failed creating customer information"  
31         });  
32     }  
33  
34     let response_body: Value = json!{  
35         "message": "Customer added successfully",  
36         "newcustomer": customer  
37     };  
38     HttpResponse::Created().json(response_body)  
39 }  
40  
41  
42 #[delete("/customer/{id}")]  
43 async fn delete_customer_by_id(id: web::Path<i32>) -> impl Responder {  
44     let customer_id: i32 = id.into_inner();  
45  
46     if customer_id == 3 {  
47         let response_body: Value = json!{  
48             "message": "Customer deleted"  
49         };  
50         HttpResponse::Ok().json(response_body)  
51     } else {  
52         return HttpResponse::NotFound().json(json!{  
53             "message": "No customer Found"  
54         });  
55     }  
56 }  
57 }
```

handler

09

bus_handler.rs

สำหรับจัดการข้อมูลของ Bus ซึ่งประกอบด้วยฟังก์ชันต่างๆ เช่น get_bus() สำหรับดึงข้อมูลรถบัสกั้งหมด, update_bus_ticket() สำหรับอัปเดตข้อมูลจำนวนตัว

10

customer_handler.rs

สำหรับจัดการข้อมูลของ Customer ซึ่งประกอบด้วยฟังก์ชันต่างๆ เช่น get_customer_by_id() สำหรับแสดงข้อมูลลูกค้าตามไอดี, add_customer() สำหรับเพิ่มข้อมูลลูกค้า, delete_customer_by_id() สำหรับลบข้อมูลลูกค้าออกจากระบบตามไอดีที่กำหนด

11

seat_handler.rs

สำหรับจัดการข้อมูลของ Seat ซึ่งประกอบด้วยฟังก์ชันต่างๆ เช่น get_seat_by_id() สำหรับแสดงข้อมูลที่นั่งตามไอดีของรถบัส, update_seat_status() สำหรับอัปเดตข้อมูลสถานะที่นั่ง

handlers/seat_handler.rs

จะเป็นไฟล์ที่ใช้สำหรับการจัดการกับข้อมูลต่างๆ ในระบบ

```
EXPLORER ... @ seat_handler.rs x
src > handlers > @ seat_handler.rs > update_seat_status
1 use crate::models::seat::Seat;
2 use actix_web::{get, put, web, HttpResponse, Responder};
3 use serde_json::json;
4 #[get("/seat/{id}")]
5 async fn get_seat_by_id(id: web::Path<i32>) -> impl Responder {
6     let busid: i32 = id.into_inner();
7
8     if busid == 1 {
9         let response_body: Seat = Seat {
10             seatid: 100,
11             seatnumber: 1,
12             status: "0".to_string(),
13             busid: busid,
14             busname: "Busbin".to_string(),
15         };
16         HttpResponse::Ok().json(response_body)
17     } else {
18         return HttpResponse::NotFound().json(json!({
19             "message": "No seat available"
20         }));
21     }
22 }
23
24 #[put("/seat/{id}")]
25 async fn update_seat_status(id: web::Path<i32>, seat: web::Json<Seat>) -> impl Responder {
26     let seatid: i32 = id.into_inner();
27     let new_status: &String = &seat.status;
28
29     if new_status != "1" && new_status != "0" {
30         return HttpResponse::BadRequest().json(json!({
31             "message": "Seat status update failed"
32         }));
33     }
34
35     if seatid == 100 {
36         let response_body: Seat = Seat {
37             seatid: seatid,
38             seatnumber: 1,
39             status: new_status.to_string(),
40             busid: 1,
41             busname: "Busbin".to_string(),
42         };
43         HttpResponse::Ok().json(response_body)
44     } else {
45         return HttpResponse::NotFound().json(json!({
46             "message": "Seat ID Invalid"
47         }));
48     }
49 } fn update_seat_status
```

handler

09

bus_handler.rs

สำหรับจัดการข้อมูลของ Bus ซึ่งประกอบด้วยฟังก์ชันต่างๆ เช่น get_bus() สำหรับดึงข้อมูลรถบัสกั้งหมด, update_bus_ticket() สำหรับอัปเดตข้อมูลจำนวนตัว

10

customer_handler.rs

สำหรับจัดการข้อมูลของ Customer ซึ่งประกอบด้วยฟังก์ชันต่างๆ เช่น get_customer_by_id() สำหรับแสดงข้อมูลลูกค้าตามไอดี, add_customer() สำหรับเพิ่มข้อมูลลูกค้า, delete_customer_by_id() สำหรับลบข้อมูลลูกค้าออกจากระบบตามไอดีที่กำหนด

11

seat_handler.rs

สำหรับจัดการข้อมูลของ Seat ซึ่งประกอบด้วยฟังก์ชันต่างๆ เช่น get_seat_by_id() สำหรับแสดงข้อมูลที่นั่งตามไอดีของรถบัส, update_seat_status() สำหรับอัปเดตข้อมูลสถานะที่นั่ง



BUS TICKET WEB

MOD.RS



mod.rs

เป็นไฟล์ที่ใช้สำหรับการรวม import ของโมดูลในโฟลเดอร์นี้ เพื่อให้ง่ายต่อการ import ในไฟล์อื่น ๆ ในโปรแกรม

```
src > routes > @ mod.rs > ...
1  pub mod bus_route;
2  pub mod customer_route;
3  pub mod seat_route;
```

```
src > models > @ mod.rs > ...
1  pub mod bus;
2  pub mod customer;
3  pub mod seat;
```

```
src > handlers > @ mod.rs > ...
1  pub mod bus_handler;
2  pub mod customer_handler;
3  pub mod seat_handler;
4
```

mod.rs

04

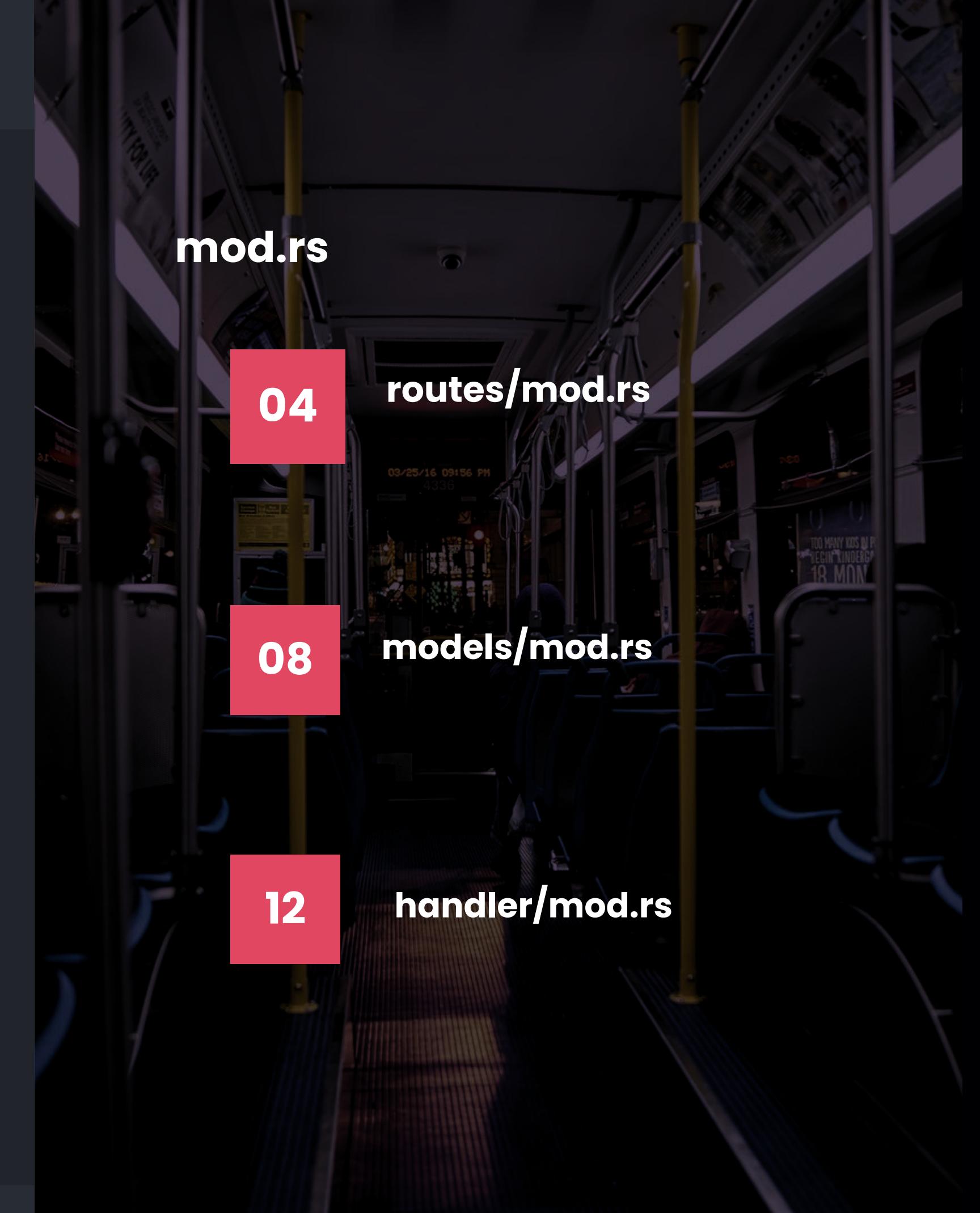
routes/mod.rs

08

models/mod.rs

12

handler/mod.rs





BUS TICKET WEB

THANK YOU

H T T P : / / 1 2 7 . 0 . 0 . 1 : 8 0 8 0 / B U S