

oooo

PRESENTATION

# LOTTERY

สลากกินแบ่ง



oooo

# ROLE OUR TEAM



JIRAPUN  
JUNRUK



CHAYAPON  
PANTASU



PUTIPONG  
RUNGTHAWEE SUP



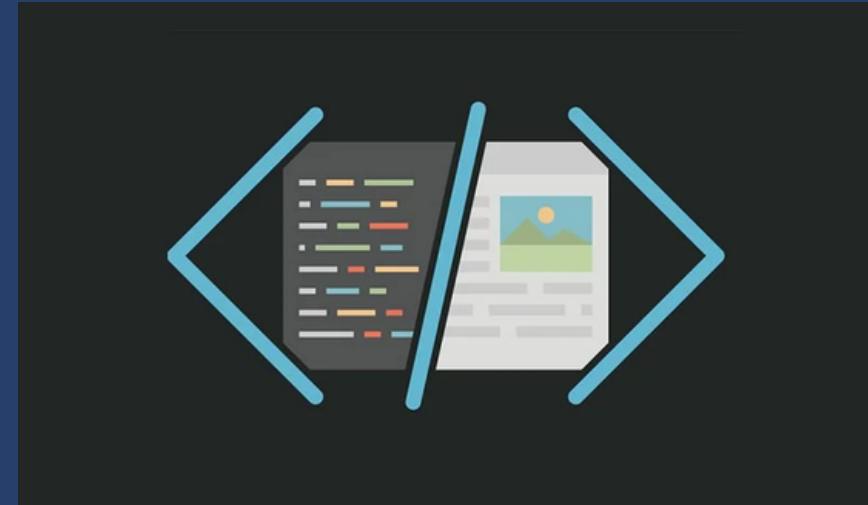
PHURIKORN  
THONGYOI

# เป้าหมายของ **LOTTERY ONLINE**

- เพื่อให้สามารถเข้าใจในการสร้างหรือการใช้ API ใน การซื้อ lottery online
- เพื่อให้สามารถออกแบบ API ให้เหมาะสมกับการใช้งาน ต่างๆได้อย่างมีประสิทธิภาพ



# ตัวอย่างการทำงาน



**FRONT END**

Front end ของเว็บ lottery online เป็นส่วนที่ผู้ใช้สามารถ และ เชื่อมต่อกับเว็บไซต์ได้โดยตรง ซึ่งประกอบไปด้วย bootstrap และ CSS ซึ่งมีหน้าที่รับข้อมูลจากผู้ใช้งานและแสดงผลข้อมูลให้กับผู้ใช้งานอย่างง่ายต่อการเข้าใจ



**BLACK END**

Backend ของเว็บ lottery online ประกอบด้วย API ส่งข้อมูล HTTP และ RUST WEP ACTIX Server ซึ่งเป็นตัวกลางในการติดต่อระหว่าง Frontend กับ Database



**DATA BASE**

Database ของเว็บ lottery online จะเป็นฐานข้อมูล MySQL สำหรับเก็บข้อมูล โดยประกอบด้วย ข้อมูลของผู้ซื้อ lottery ข้อมูลการถูกรางวัล

# DATABASE MYSQL STRUCTURE

#	Name	Type
1	admin_id	int(64)
2	F_Name	varchar(32)
3	L_Name	varchar(32)
4	Username	varchar(16)
5	Password	text

## admin

เก็บข้อมูลของ admin

admin\_id : ไอดีของ Admin

F\_Name : ชื่อของ Admin

L\_Name : นามสกุลของ Admin

Username : ชื่อผู้ใช้งาน Admin

Password : รหัสผ่านของ Admin (มีการเข้ารหัส)

#	Name	Type
1	lottery_id	int(100)
2	lottery_number	varchar(6)
3	lottery_status	varchar(9)
4	Datetime	datetime

## lottery

เก็บข้อมูลของ lottery

lottery\_id : ไอดีของ lottery

lottery\_number : หมายเลข lottery

lottery\_status : สถานะของ lottery

Datetime : วันเวลาที่ lottery ถูกเพิ่มเข้ามาในระบบ

#	Name	Type
1	id	int(11)
2	user_id	int(64)
3	lottery_id	int(100)

## basket

เก็บข้อมูลของ basket

id : ไอดีของ ตะกร้า

user\_id : ไอดีของ User

lottery\_id : ไอดีของ lottery

# DATABASE MYSQL STRUCTURE

#	Name	Type
1	reward_id 🔑	int(32)
2	reward_number	varchar(6)
3	Datetime	datetime

## reward

เก็บข้อมูลของ reward

reward\_id : ไอดีของ รางวัล

reward\_number : หมายเลขรางวัล

Datetime : วันเวลาที่ รางวัล ถูกเพิ่มเข้ามาในระบบ

#	Name	Type
1	id 🔑	int(11)
2	user_id	int(64)
3	lottery_number	varchar(6)
4	Datetime	datetime

## user\_history

เก็บข้อมูล history ของ user

id : ไอดีของ history

user\_id : ไอดีของ User

lottery\_number : หมายเลข lottery

Datetime : วันเวลาที่ lottery ถูกซื้อ

#	Name	Type
1	user_id 🔑	int(64)
2	F_Name	varchar(32)
3	L_Name	varchar(32)
4	Username	varchar(16)
5	Password	text

## user

เก็บข้อมูลของ user

user\_id : ไอดีของ User

F\_Name : ชื่อของ User

L\_Name : นามสกุลของ User

Username : ชื่อผู้ใช้งานของ User

Password : รหัสผ่านของ User  
(มีการเข้ารหัส)

# **FILE STRUCTURE**

## **Controller**

lottery.rs  
basket.rs  
customer.rs  
prize.rs  
admin\_prize  
admin\_lottery

## **Routes**

admin\_routes.rs  
customer\_routes.rs  
basket\_routes.rs  
lottery\_routes.rs  
prize\_routes.rs

## **models**

admin\_lottery\_model.rs  
basket\_model.rs  
lottery\_model.rs  
prize\_model.rs  
admin\_prize\_model.rs  
customer\_model.rs  
product.rs



**MAIN.RS**

# Main.rs เป็นไฟล์ที่เริ่มต้นการทำงานของแอปพลิเคชันโดยให้ค่าโดยอัตโนมัติ

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** main.rs - Cloud-Project-Kmutnb - Visual Studio Code.
- Explorer Bar (Left):** Shows the project structure under "CLOUD-PROJECT-KMUTNB".
  - src: db, config, controllers, models, public, routes, main.rs (selected).
  - target, .gitignore, Cargo.lock, Cargo.toml, README.md.
- Editor Area (Main View):** Displays the content of the main.rs file.

```
src > main.rs > ...
4
5 mod controllers;
6 mod models;
7 mod config;
8 pub mod routes;
9 use crate::routes::*;

10
11 #[actix_web::main]
12 ▶ Run | Debug
13 async fn main() -> std::io::Result<()> {
14
15     env_logger::Builder::from_env(Env::default().default_filter_or("debug")).init();
16
17     HttpServer::new(|| {
18         App::new()
19             .wrap(middleware::Logger::default())
20             .configure(lottery_routes::config)
21             .configure(basket_routes::config)
22             .configure(prize_routes::config)
23             .configure(customer_routes::config)
24             .configure(admin_routes::config)
25             .default_service(web::route().to(not_found)) // กำหนด handler 404 ด้วย middleware
26     })
27
28     .bind("127.0.0.1:3000")?
29     .run()
30     .await
31 }
32
33
34 async fn not_found() -> Result<NamedFile> {
35     Ok(NamedFile::open(path: "src/public/error404.html")?)
36 }
37
38
```
- Right Side Panels:** Includes the Taskbar, Output, and Search panels.



**ROUTES**

# Routes ໃຫ້ກຳບົດເສັນທາງ API ສໍາຮັບກາຮຈັດກາຮຂ້ອມູລ

The screenshot shows a Visual Studio Code interface with four tabs open, each containing Rust code for configuring Actix-Web service endpoints:

- customer\_routes.rs**: Configures a service endpoint for getting a prize.
- prize\_routes.rs**: Configures a service endpoint for getting a prize.
- admin\_routes.rs**: Configures multiple service endpoints for managing prizes and lotteries.
- lottery\_routes.rs**: Configures service endpoints for getting, putting, and deleting lotteries.

The code in each file uses the `actix_web::web` crate and its `controllers` module to define routes and services. The `config` function takes a mutable reference to a `ServiceConfig` and adds various service endpoints using the `.service` method.

```
File Edit Selection View Go Run Terminal Help
lottery_routes.rs - Cloud-Project-Kmutnb - Visual Studio Code
customer_routes.rs prize_routes.rs admin_routes.rs lottery_routes.rs
src > routes > customer_routes.rs > config
src > routes > prize_routes.rs > config
src > routes > admin_routes.rs > config
src > routes > lottery_routes.rs > ...
use actix_web::web;
use crate::controllers::prize::{get_prize};
use actix_web::web;
use crate::controllers::admin::*;

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

pub fn config(cfg: &mut web::ServiceConfig) {
    cfg &mut ServiceConfig
        .service(factory: admin_prize::get_admin_prize) &mut ServiceConfig
        .service(factory: admin_prize::post_admin_prize) &mut ServiceConfig
        .service(factory: admin_lottery::get_admin_lottery) &mut ServiceConfig
        .service(factory: admin_lottery::post_admin_lottery) &mut ServiceConfig
        .service(factory: admin_lottery::delete_admin_lottery);
}

pub fn config(cfg: &mut web::ServiceConfig) {
    cfg &mut ServiceConfig
        .service(factory: get_basket) &mut ServiceConfig
        .service(factory: post_basket) &mut ServiceConfig
        .service(factory: delete_basket) &mut ServiceConfig
        .service(factory: get_basket_verification);
}

use actix_web::web;
use crate::controllers::lottery::{get_lottery, put_lottery};

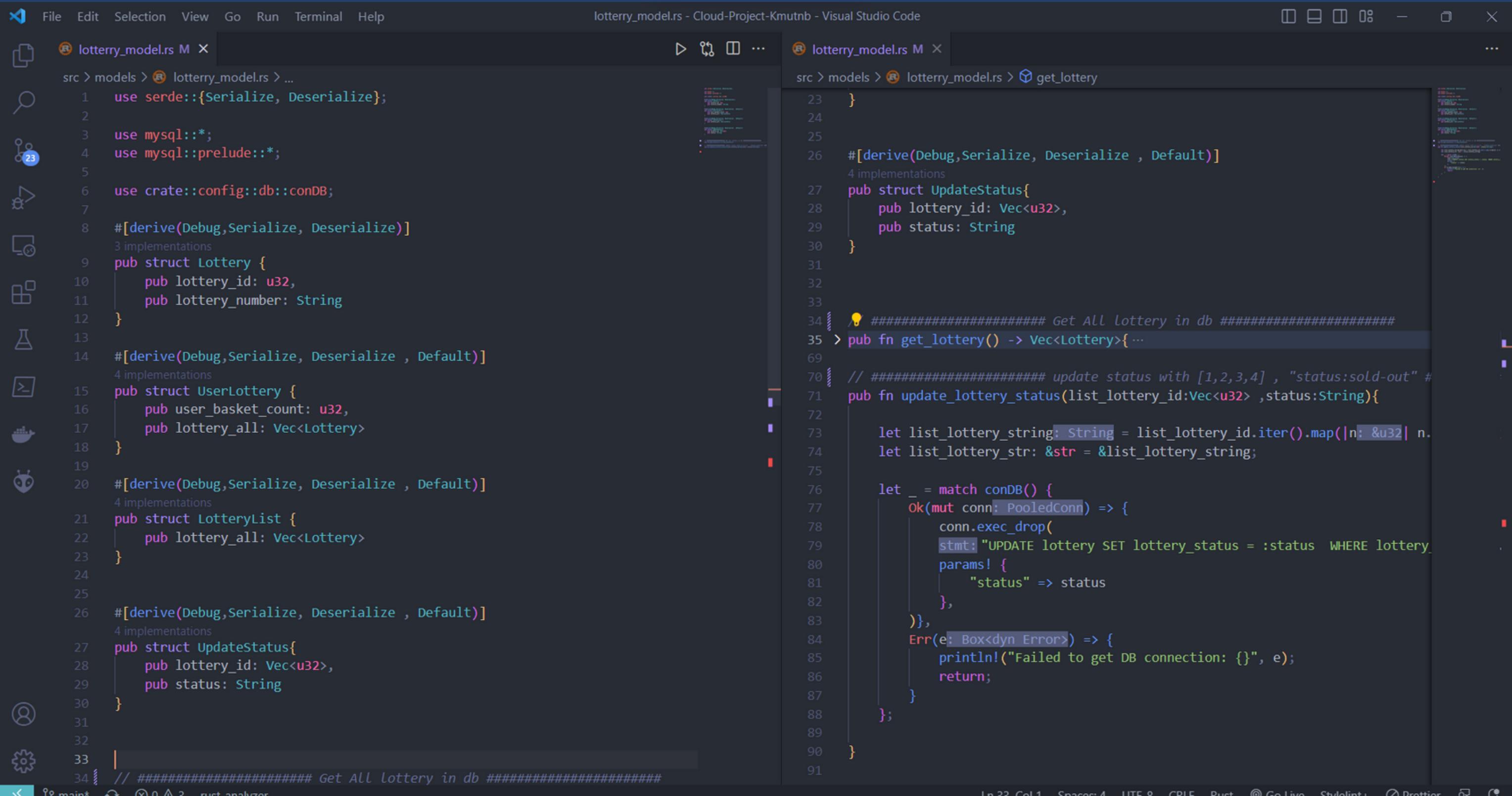
pub fn config(cfg: &mut web::ServiceConfig) {
    cfg &mut ServiceConfig
        .service(factory: get_lottery) &mut ServiceConfig
        .service(factory: put_lottery);
}
```



**MODELS**

# Model

## lottery\_model.rs



The screenshot shows two tabs of the same file, `lottery_model.rs`, in Visual Studio Code. The left tab displays the full file content, while the right tab shows a specific section of the code.

```
File Edit Selection View Go Run Terminal Help
lottery_model.rs - Cloud-Project-Kmutnb - Visual Studio Code
lottery_model.rs M X
src > models > lottery_model.rs > ...
1 use serde::{Serialize, Deserialize};
2
3 use mysql::*;
4 use mysql::prelude::*;
5
6 use crate::config::db::conDB;
7
8 #[derive(Debug, Serialize, Deserialize)]
9 3 implementations
10 pub struct Lottery {
11     pub lottery_id: u32,
12     pub lottery_number: String
13 }
14
15 #[derive(Debug, Serialize, Deserialize, Default)]
16 4 implementations
17 pub struct UserLottery {
18     pub user_basket_count: u32,
19     pub lottery_all: Vec<Lottery>
20 }
21
22 #[derive(Debug, Serialize, Deserialize, Default)]
23 4 implementations
24 pub struct LotteryList {
25     pub lottery_all: Vec<Lottery>
26 }
27
28 #[derive(Debug, Serialize, Deserialize, Default)]
29 pub struct UpdateStatus{
30     pub lottery_id: Vec<u32>,
31     pub status: String
32 }
33
34 // ##### Get All lottery in db #####
35
36 #[derive(Debug, Serialize, Deserialize, Default)]
37 pub fn get_lottery() -> Vec<Lottery>{
38
39     // ##### update status with [1,2,3,4] , "status:sold-out" #
40     pub fn update_lottery_status(list_lottery_id:Vec<u32>,status:String){
41
42         let list_lottery_string: String = list_lottery_id.iter().map(|n: &u32| n.to_string()).collect();
43         let list_lottery_str: &str = &list_lottery_string;
44
45         let _ = match conDB() {
46             Ok(mut conn: PooledConn) => {
47                 conn.exec_drop(
48                     stmt: "UPDATE lottery SET lottery_status = :status WHERE lottery_id = :id",
49                     params! {
50                         "status" => status
51                     },
52                 ),
53                 Err(e: Box<dyn Error>) => {
54                     println!("Failed to get DB connection: {}", e);
55                     return;
56                 }
57             };
58         }
59
60     }
61
62 }
63
64 // ##### Get All lottery in db #####
65
66 }
```

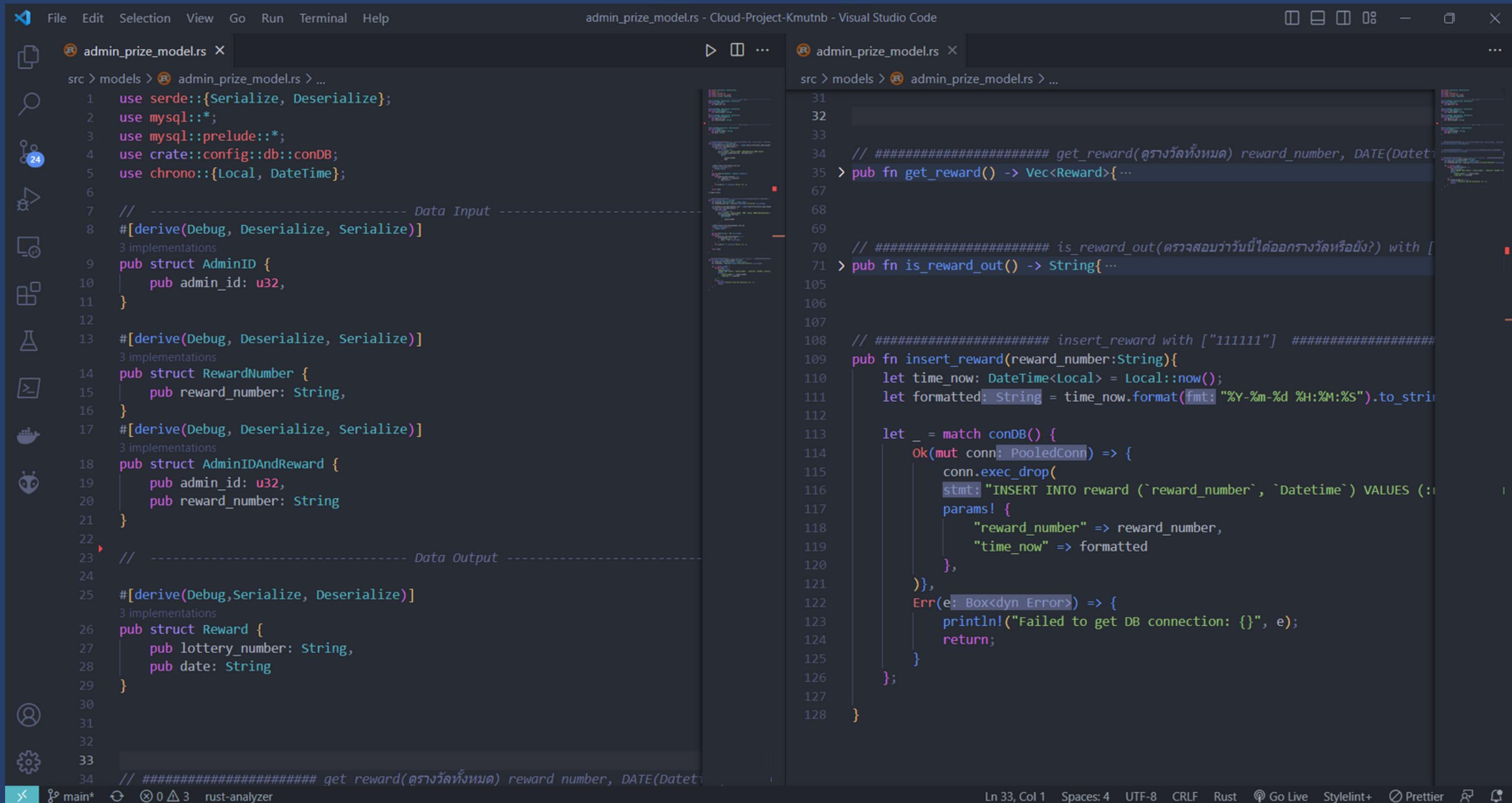
The code defines several structs: `Lottery`, `UserLottery`, `LotteryList`, and `UpdateStatus`. It includes methods `get_lottery()` and `update_lottery_status()`. The `update_lottery_status` method uses a database connection to update the `lottery_status` field for multiple lottery entries.

# Model

# basket\_model.rs

# Model

## admin\_prize\_model.rs



The screenshot shows two side-by-side tabs in Visual Studio Code, both displaying the same file: `admin_prize_model.rs`. The file contains Rust code for managing rewards.

```
use serde::Serialize, Deserialize;
use mysql::*;
use mysql::prelude::*;
use crate::config::db::conDB;
use chrono::Local, DateTime;

// ----- Data Input -----
#[derive(Debug, Deserialize, Serialize)]
impl AdminID {
    pub struct AdminID {
        pub admin_id: u32,
    }
}

#[derive(Debug, Deserialize, Serialize)]
impl RewardNumber {
    pub struct RewardNumber {
        pub reward_number: String,
    }
}

#[derive(Debug, Deserialize, Serialize)]
impl AdminIDAndReward {
    pub struct AdminIDAndReward {
        pub admin_id: u32,
        pub reward_number: String
    }
}

// ----- Data Output -----
#[derive(Debug, Serialize, Deserialize)]
impl Reward {
    pub struct Reward {
        pub lottery_number: String,
        pub date: String
    }
}

// ##### get_reward(ดูรางวัลทั้งหมด) reward number, DATE(Datetime)
// ##### is_reward_out(ตรวจสอบว่าวันนี้ได้ออกรางวัลหรือยัง?) with [
// ##### insert_reward with ["111111"] #####
pub fn get_reward() -> Vec<Reward>{
    // ...
}

pub fn is_reward_out() -> String{
    // ...
}

pub fn insert_reward(reward_number:String){
    let time_now: DateTime<Local> = Local::now();
    let formatted: String = time_now.format(fmt: "%Y-%m-%d %H:%M:%S").to_string();

    let _ = match conDB() {
        Ok(mut conn: PooledConn) => {
            conn.exec_drop(
                stmt: "INSERT INTO reward (`reward_number`, `Datetime`) VALUES (:reward_number, :time_now)",
                params! {
                    "reward_number" => reward_number,
                    "time_now" => formatted
                },
            ),
            Err(e: Box<dyn Error>) => {
                println!("Failed to get DB connection: {}", e);
                return;
            }
        };
    }
}
```

The code defines several structs: `AdminID`, `RewardNumber`, `AdminIDAndReward`, and `Reward`. It includes methods for getting all rewards, checking if a reward has been issued today, and inserting a new reward into the database. The database connection is managed using the `conDB()` function.



**CONTROLER**

# Controller

## lottery.rs

The screenshot shows the Visual Studio Code interface with the file `lottery.rs` open. The code implements a controller for a lottery system using the actix-web framework in Rust. It defines two routes: `/lottery` (GET) and `/lottery` (PUT).

```
File Edit Selection View Go Run Terminal Help lottery.rs - Cloud-Project-Kmutnb - Visual Studio Code lottery.rs M X src > controllers > lottery.rs > ... 1 use actix_web::{web, get, put, Responder, HttpResponse}; 2 3 4 use crate::models::lottery_model::*;

5 use crate::models::basket_model::get_user_count_basket;
6 7 #[get("/lottery")]
8 async fn get_lottery(user_id: web::Json<GetUserData>) -> impl Responder {
9
10     if user_id.user_id == 1{
11
12         let lottery_item: Vec<Lottery> = lottery_model::get_lottery();
13
14         let res: UserLottery = UserLottery{
15             user_basket_count:get_user_count_basket(user_id.user_id.try_into().unwrap()),
16             lottery_all:lottery_item
17         };
18
19
20         return HttpResponse::ok().json(res);
21
22     }else{
23
24         return HttpResponse::Unauthorized().json("Error");
25     }
26
27
28 }
29
30
31
32 #[put("/lottery")]
33 async fn put_lottery(lottery: web::Json<UpdateStatus>) -> impl Responder { ...
34
35
36
37
38
39
40
41 }
```

The code uses the `actix-web` library for handling HTTP requests. It imports necessary modules from the `crate::models` namespace, specifically `lottery_model` and `basket_model`. The `get` route returns a JSON response containing the lottery items and the user's basket count. The `put` route is currently a placeholder.

# Controller

## basket.rs

```
File Edit Selection View Go Run Terminal Help
basket.rs - Backend - Visual Studio Code
src > controllers > basket.rs > delete_basket
40 } fn post_basket
41
42
43 #[get("/basket")] //[]
44 async fn get_basket(user_id: web::Json<GetUserData>) -> impl Responder {
45     // [1] ตรวจสอบ userID
46     if user_id.user_id == 1{
47         // [2] res select lottery ทั้งหมดในตะกร้า user
48         let lottery_item: Vec<Lottery> = get_user_lottery_number();
49
50         HttpResponse::ok().json(lottery_item)
51
52     }else{
53         HttpResponse::Unauthorized().json("Error")
54     }
55 }
56
57 }
58
59
60
61 #[delete("/basket")] //[]
62 async fn delete_basket(lottery: web::Json<LotteryIDwithUserID>) -> impl Responder {
63
64     // [1] ตรวจสอบ userID
65     // debug!("TEST {:?}", &lottery);
66     // lottery.user_id , lottery.lottery.lottery_id
67     delete_user_basket(lottery.user_id , lottery.lottery_id);
68     // [2] delete lottery by id ในตะกร้า
69     return HttpResponse::ok().json("succeed");
70 }
71
72
73 #[get("/basket/verification")] //[]
74 async fn get_basket_verification(lottery: web::Json<LotteryArrayID>) -> impl Responder {
75
76     // [1] ตรวจสอบ userID
77 }
```

Ln 70, Col 2 Spaces: 4 UTF-8 CRLF Rust Go Live Prettier

# Controller

## admin\_lottery.rs

```
File Edit Selection View Go Run Terminal Help admin_lottery.rs - Cloud-Project-Kmutnb - Visual Studio Code admin_lottery.rs M X src > controllers > admin > admin_lottery.rs > post_admin_lottery 1 use actix_web::{web, get, post, delete, Responder, HttpResponse}; 2 3 use log::debug; 4 5 6 use crate::models::admin_lottery_model::*; 7 8 use rand::seq::SliceRandom; 9 10 11 #[get("/admin/lottery")] //[] 12 async fn get_admin_lottery(admin: web::Json<AdminID>) -> impl Responder { 13 14     if admin.admin_id == 1{ 15         let all_date: Vec<Date> = get_date_unique(); 16         let mut lottery_detail:Vec<LotteryDateCount> = Default::default(); 17         for i in all_date{ 18             lottery_detail.push(get_lottery(i.date)); 19         } 20         return HttpResponse::Ok().json(lottery_detail); 21     }else{ 22         return HttpResponse::Unauthorized().json("Error"); 23     } 24 } 25 26 27 } 28 29 #[post("/admin/lottery")] //[] 30 async fn post_admin_lottery(admin: web::Json<AdminIDCount>) -> impl Responder { 31     let admin: AdminIDCount = admin.into_inner(); 32     if admin.admin_id == 1{ 33         let number: Vec<String> = random_numbers(admin.lottery_count as usize); 34         for i in 0..admin.lottery_count{ 35             debug!("Test Get {:?}", "1"); 36             insert_lottery(number[i as usize].to_string()); 37         } 38     } 39 } 40 41 } 42 43 #[delete("/admin/lottery")] //[] 44 async fn delete_admin_lottery(admin: web::Json<AdminIdDate>) -> impl Responder { 45     let admin: AdminIdDate = admin.into_inner(); 46     if admin.admin_id == 1{ 47         delete_lottery_by_date(admin.date); 48         return HttpResponse::Ok().json("lottery_detail"); 49     }else{ 50         return HttpResponse::Unauthorized().json("Error"); 51     } 52 } 53 54 } 55 56 } 57 58 } 59 60 } 61 62 } 63 64 }
```

The screenshot shows two side-by-side code editors in Visual Studio Code. Both editors have the same file open: `admin_lottery.rs`. The left editor displays the `get_admin_lottery` function, which retrieves lottery details for an admin with ID 1. It uses `get_date_unique` to get unique dates and `insert_lottery` to store the results. The right editor displays the `post_admin_lottery` and `delete_admin_lottery` functions. The `post_admin_lottery` function generates random numbers and stores them in the lottery database. The `delete_admin_lottery` function removes lottery entries by date. The code is written in Rust using the `actix-web` framework.