The Faculty of Information and Communication Technology

Mahidol University

Project Phase II

Miss Supithcha	Jongphoemwatthanaphon	6488045
Miss Kanita	Karunkittikun	6488049
Miss Sasasuang	Pattanakitjaroenchai	6488052
Miss Nisakorn	Ngaosri	6488226

ITCS212 Web Programming

Dr. Wudhichart Sawangphol

Dr. Jidapa Kraisanka

Dr. Pisit Praiwattana

April 25, 2023

Table of contents

Project Overview

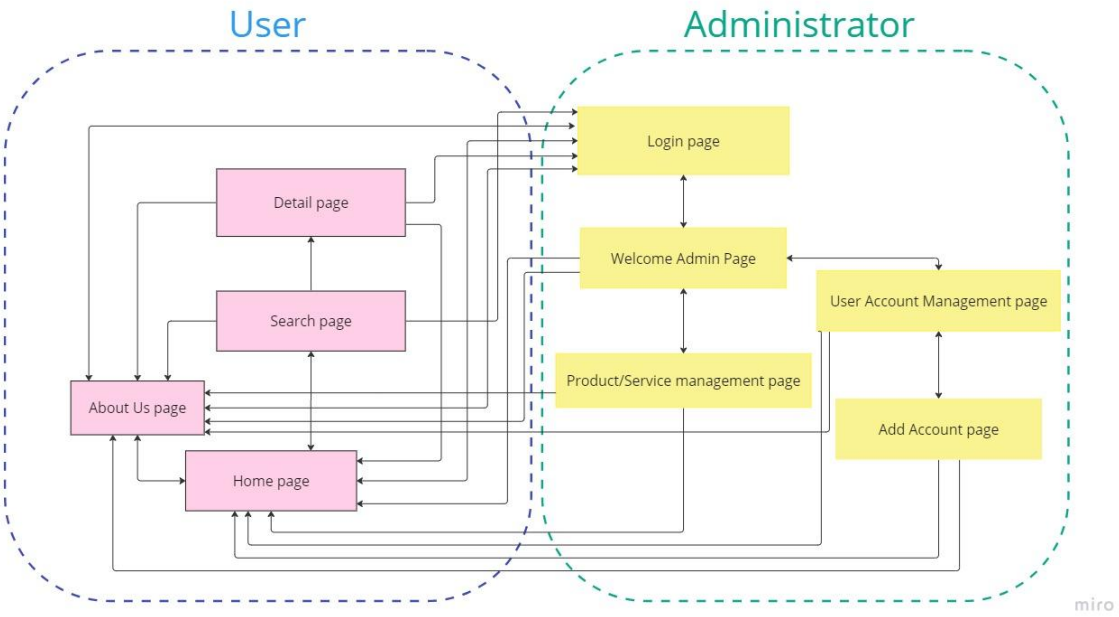
This project is part of the ITCS212 Web Programming course in which we have created and developed a web application based on the knowledge we learned in class. Our project has been divided into two main components.

In Phase I of our project, we have involved the design of each webpage using HTML, CSS, and JavaScript, focusing on creating a website related to the Thailand railway. Our website has been divided into two primary sections: one for users and one for administrators. Users can access the Homepage which is the main page of our website, the Search page, the Search Results page, and the About Us page. On the other hand, administrators need to log in successfully through the Login page to access the Welcome Admin page, Product Management page, User Account Management page, and Add Account page.

In phase II of our project, we developed a web service using Node.js for our website. To support the web service, we created three databases that store administration information, administrator login information, and ticket information. Using the data from these databases, we implemented specific functions that enable administrators to search, view, edit, and delete both ticket and administrator information. These functions are protected by authentication by using a token that only allows authorized administrators to access them. Additionally, we also integrated WeatherAPI.com on the homepage to provide real-time weather data and geolocation information for specific cities. We achieved this by connecting our web application to the public web service using Fetch API. The front-end of our website's user interface was implemented on localhost:3030, while the back-end web service was implemented on localhost:3000.

Presentation video link:
https://drive.google.com/file/d/14BzooSZOAFjI62NCD18X18YWLteO9ccn/view?usp=sharing

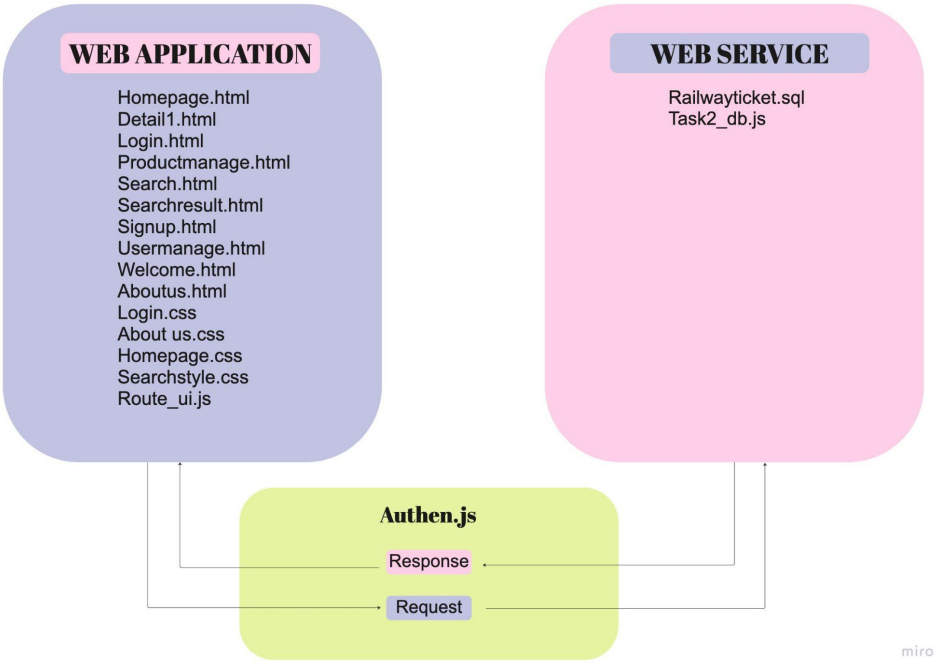
Navigation Diagram:



The navigation diagram is organized into two primary categories: users and administrators. The user category consists of three main pages: a Home page, a Search page, and an About Us page. On the other hand, the administrator comprises a Login page, a Welcome Admin page, a Product Management page, a User Account Management page, and an Add account page.

The navigation bar is present on every page. However, the pages that allow only administrators to access include the Welcome Admin page, Product Management page, User Account Management page, and Add Account page.

Web service workflow



Detail of web application and code

Source code directory files for a web application:

HTML files

- Homepage.html
- detail1.html
- login.html
- productmanage.html
- search.html
- signup.html
- usermanage.html
- welcome.html
- aboutus.html

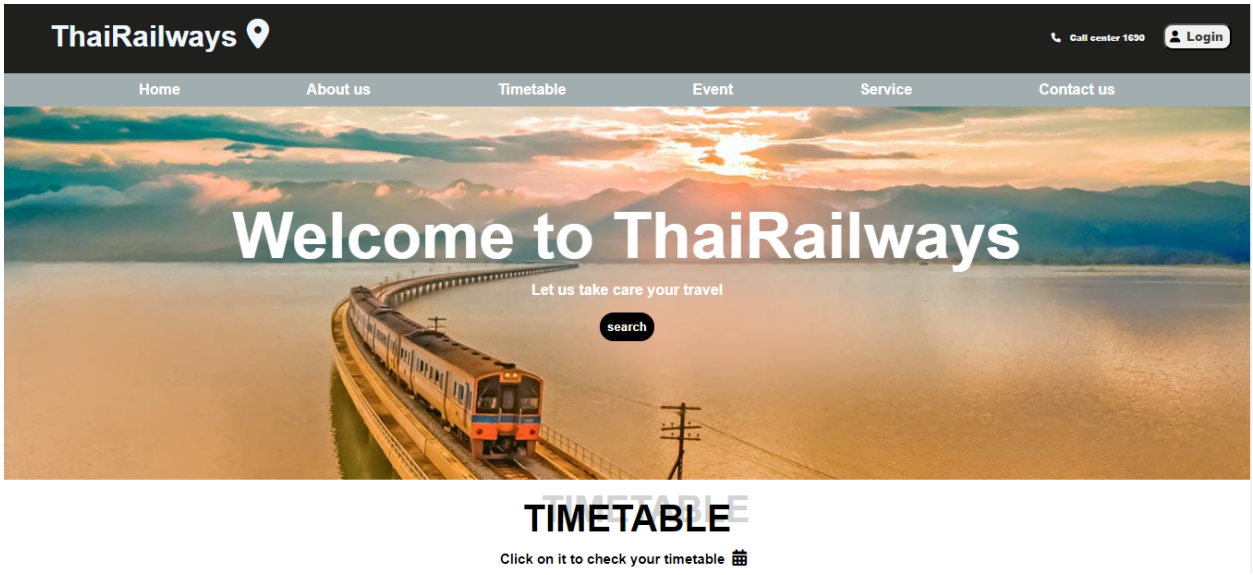
CSS files

- login.css
- about us.css
- Homepage.css
- Searchstyle.css

JS files

- route_ui.js
- authen.js

Home page:



URL: http://localhost:3030/

The header part includes the name and a button for accessing the Login page that can link to the Login page. This page navigation bar features links to the Home, About Us, Timetable, Event, Service, and Contact Us pages. Clicking on the "About Us" button will direct you to the About Us page.

In the section part, we provide a search button for linking to the “Search” page.

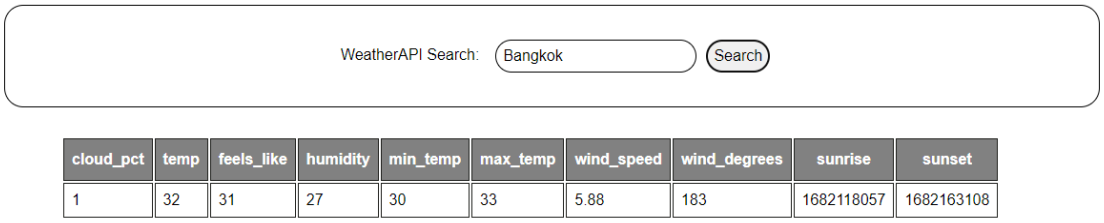
The main content of the Home page contains Home, Timetable, Event, Service, and Contact Us content. For user convenience, you can simply click on the content header on the navigation bar features links, which use the [Follow link] to access content on the Home page.

- “TIME TABLE” (e.g. the Northern line, Northeastern line, Southern line, Eastern line, Commuter line, Wongwian Yai line, Ban Laem line, and Tourist train line). URL when click: http://localhost:3030/#Timetable
- “NEW EVENTS” illustrates the information about the new event. URL when click: http://localhost:3030/#Event
- “OUR SERVICE” also illustrates the information about the service of ticket purchasing. URL when click: http://localhost:3030/#service

- “CONTACT US” (e.g. Contact us, Get help, and Follow us). URL when click: http://localhost:3030/#contact

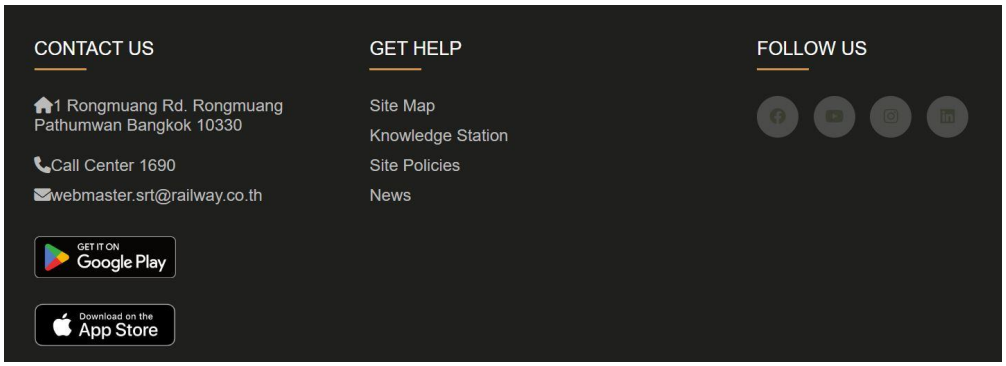
In addition, we incorporated a new feature that allows users to search for weather information using WeatherAPI. This API provides real-time weather data, including temperature, humidity, wind speed, wind degrees, sunrise, and sunset information.
Reference source for public API: https://rapidapi.com/weatherapi/api/weatherapi-com/

Example WeatherAPI:

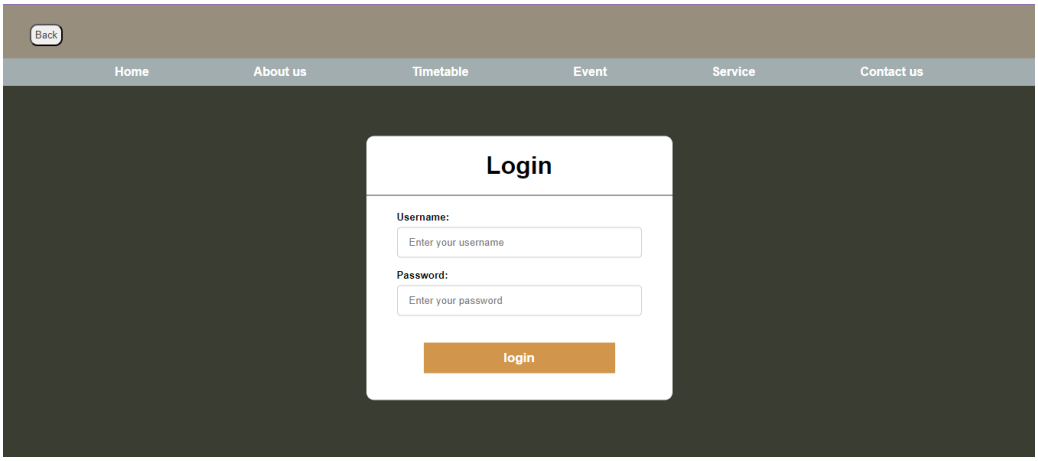


cloud_pct	temp	feels_like	humidity	min_temp	max_temp	wind_speed	wind_degrees	sunrise	sunset
1	32	31	27	30	33	5.88	183	1682118057	1682163108

The website's footer section contains details such as the address, call number, and various icons.

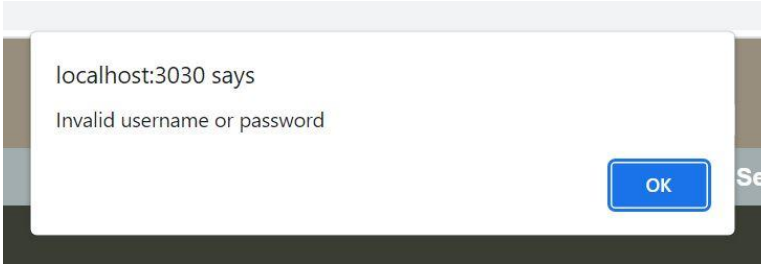


Login page:



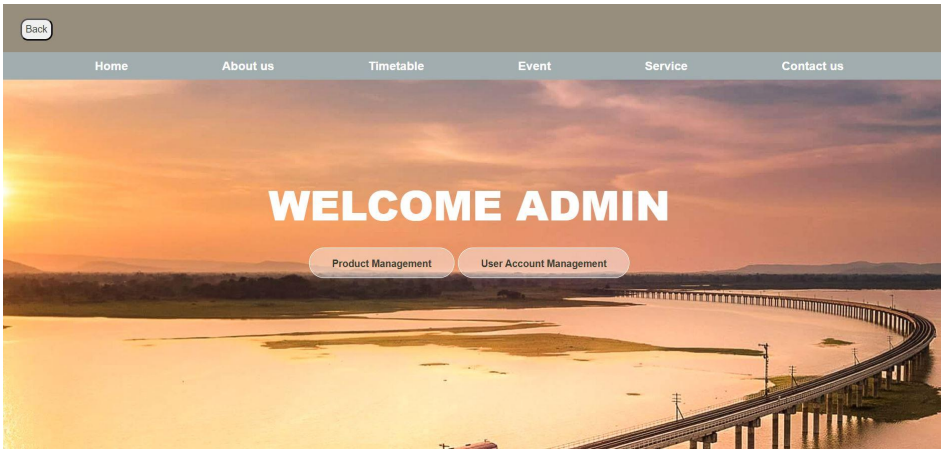
URL: http://localhost:3030/login

The login page includes the 2 main elements: Username, a field where input the username associated with their account. Password is a blank where the user enters their password for the account profile. Upon clicking the Login button, the system verifies the entered information by comparing it with the database. If the entered information is correct and matches the data in the database, the user is redirected to the "Welcome Admin" page.



In case the username or password does not match the data in the database, a warning message will appear as depicted in the image.

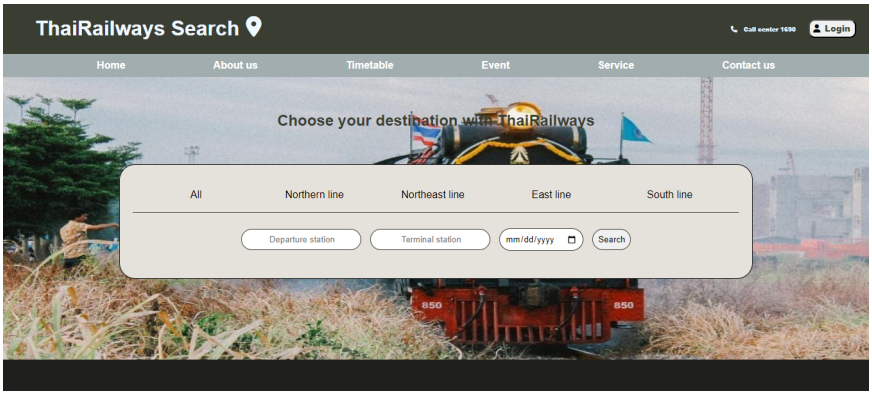
Welcome Admin page:



URL: http://localhost:3030/welcome

The purpose of this page is to navigate administrators to select the next page. This page offers two options for selection: the Product/Service Management page and the User Account Management page. When clicking on the Product/Service Management button, the system will automatically redirect administrators to the Product/Service Management page. Similarly, if administrators click the User Account Management button, the system will redirect administrators to the User Account Management page.

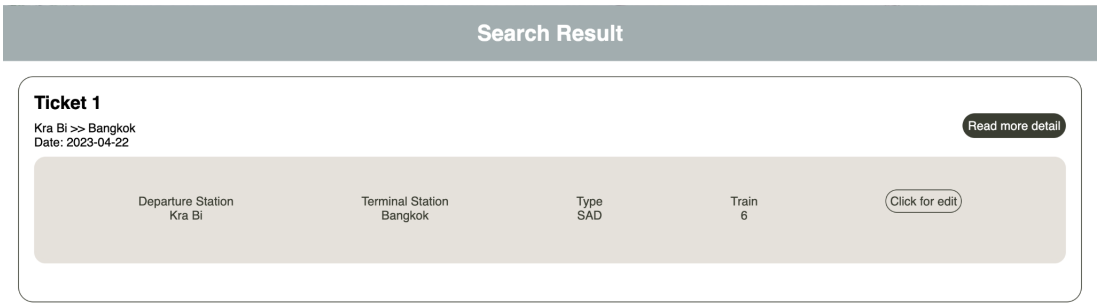
Search page:



URL: http://localhost:3030/search

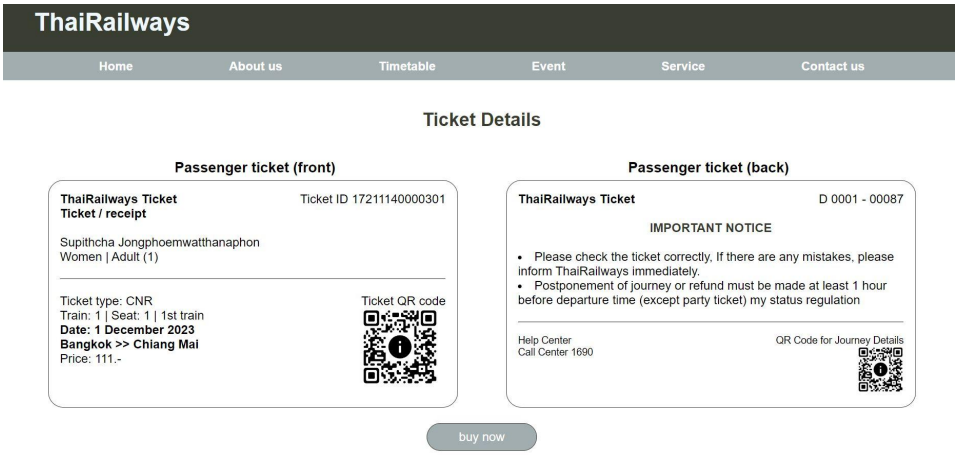
The Search page provides three search criteria, including Departure station, Terminal station, and available date. Upon pressing the Search button, all ticket information will be displayed. The user can search using any one , two, or all of the criteria for the search, and the search results will be shown accordingly.

Search result example:



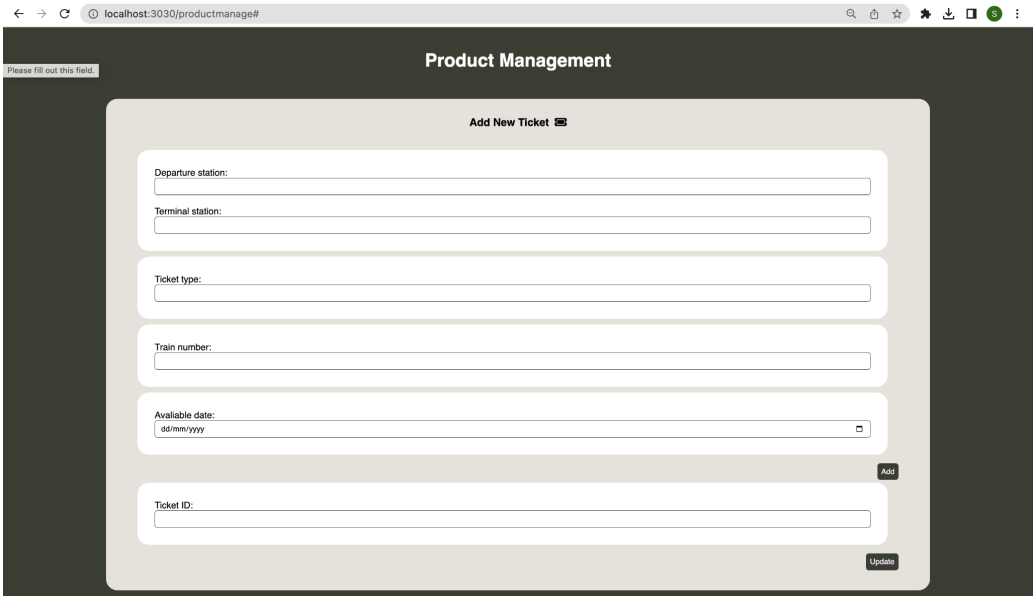
In case the user tries to search for ticket information on the webpage, but the information is not present in the database, a message stating "no result" will be shown.

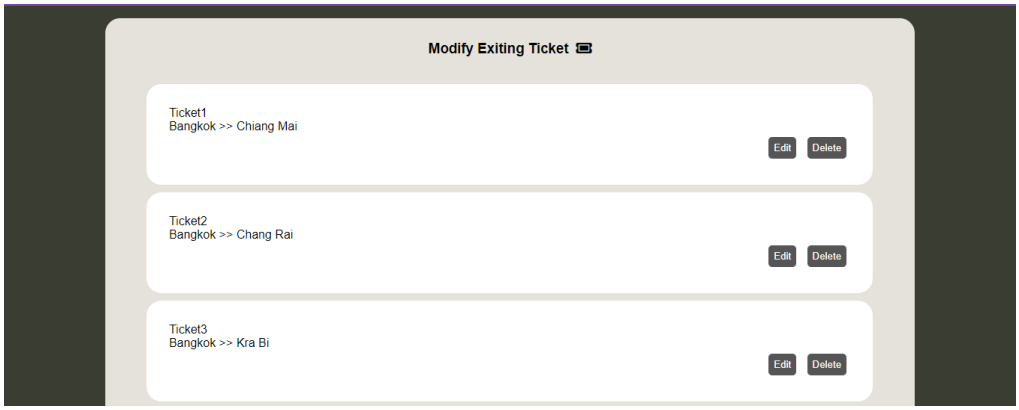
In case the user tries to search for ticket information on the webpage, but the information is not present in the database, a message stating "no result" will be shown.



If the user presses the Read more details button, the information of that ticket will be shown. [note: The current design of the project has a limitation where, upon clicking the "Read more details" button, the user is directed to the same file, "detail1.html," regardless of the specific product being viewed.]

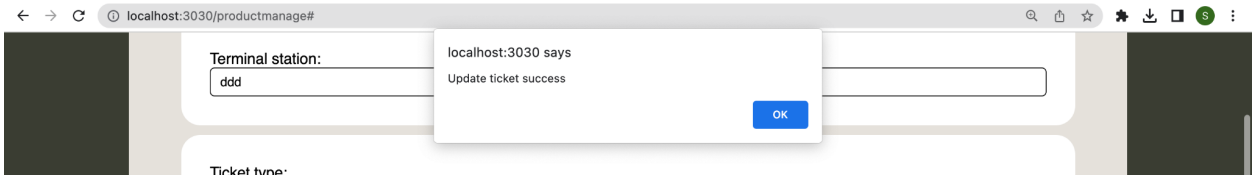
Product/Services management page:





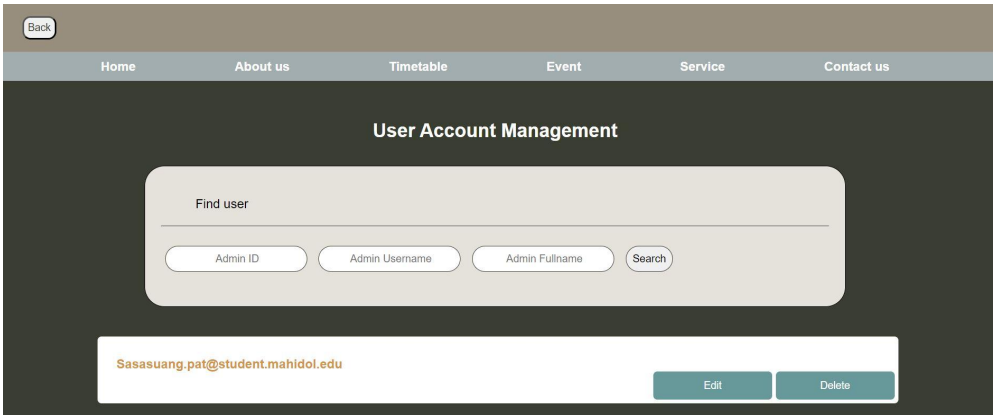
URL: http://localhost:3030/productmanage

On this page, the user needs to fill out the information for adding a new ticket as follows: Departure station, Terminal station, Ticket type, Train number, and Available date. In case the user intends to update the ticket information, they must provide the new details and Ticket ID that they want to update. Additionally, there is a section for modifying existing tickets, which is accessible to administrators. If they wish to edit the ticket information, they can click on the Edit button, which reload them to the Add new ticket block. Furthermore, if the administrators want to delete a ticket, they can press the Delete button to remove it from the system.



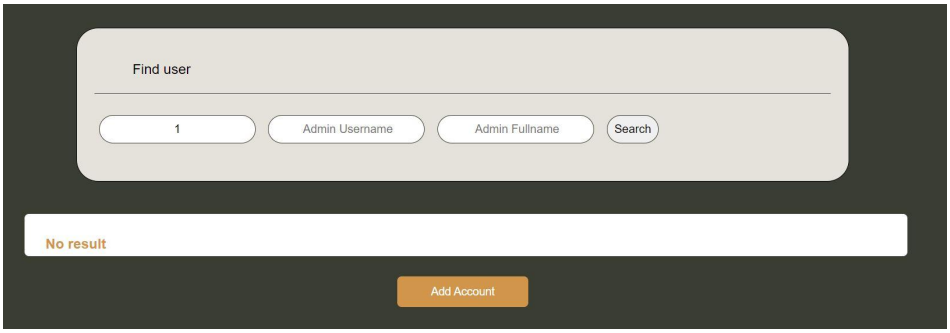
If the ticket's information has been updated, there will be a notification as in the picture. Additionally, the updated ticket information will be updated in the database. The updated ticket will be visible in the search for all ticket results.

Admin Account Management page:



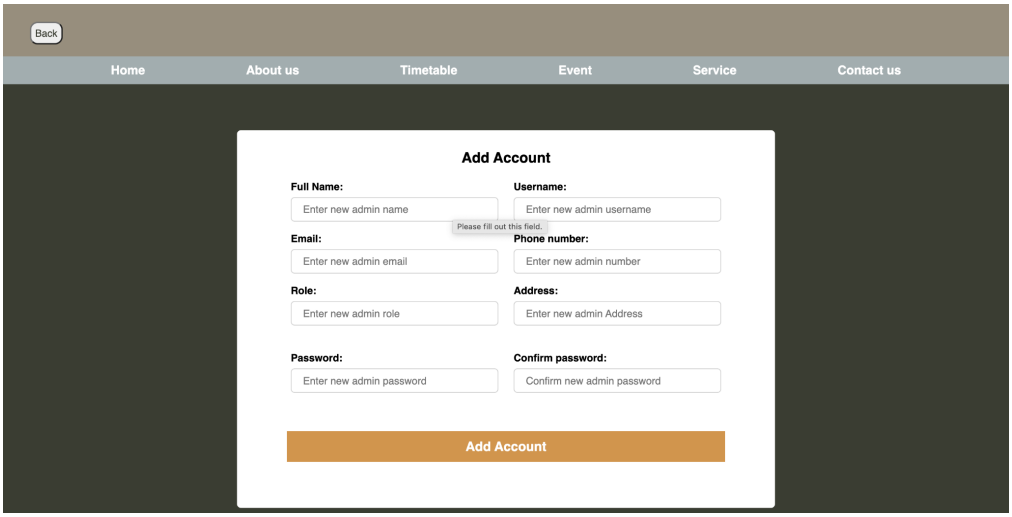
URL: http://localhost:3030/usermanage

The function of the User Account Management page is to find users and manage user accounts within the platform. This page provides three search criteria, including admin ID, admin username, and admin full name. Upon pressing the Search button, all administrator information will be displayed. The user can search using any one, two, or all of the criteria for the search, and the search results will be shown accordingly. In addition to displaying user information, it provides administrators to create, modify, or remove user accounts. For adding or updating a new account, can be added or updated from the Add/Update account button at the bottom of the page. After clicking the button, it will link to the Add/Update account pages for storing or updating new user information. Moreover, clicking the “Edit” button also links to the Add/Update account pages. On the contrary, the Delete button can be utilized for removing an admin account. However, clicking the “Delete” button will remove that specific user account.



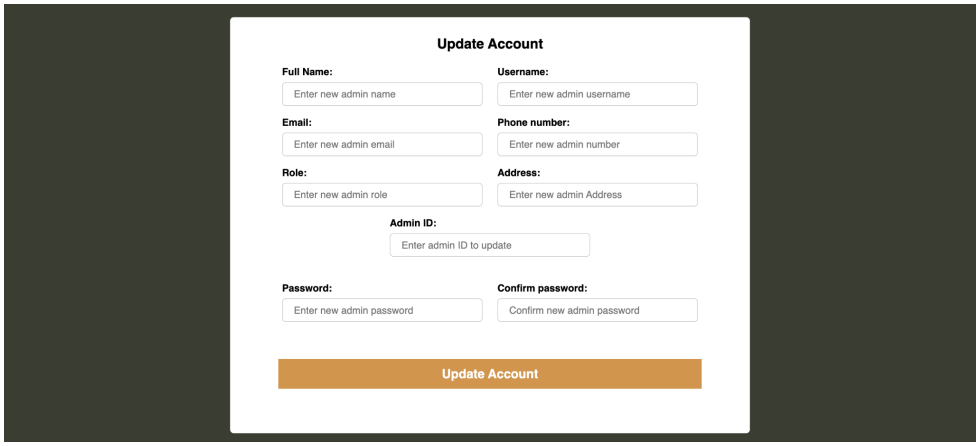
If the admin attempts to search for information on admin on the webpage but is unable to find it due to the absence of that admin's information in the database, a message indicating no result will be displayed.

Add/Update Account page:

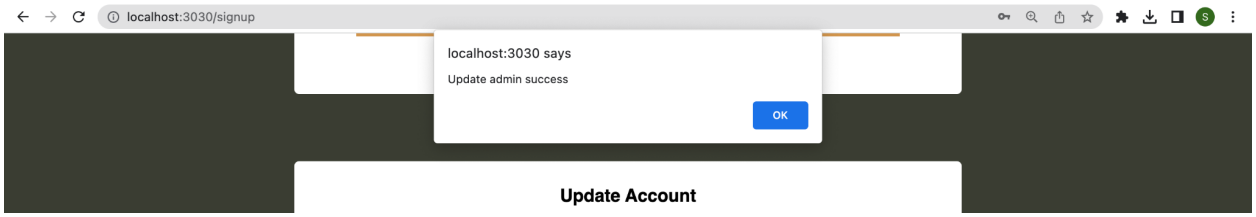


URL: http://localhost:3030/signup

The purpose of the add account block is to collect and save the information of a new administrator. The data collected will include the administrator's name, username, email address, phone number, role, and address. After the user completes the information, the user can click the Add button. It will enable the new user to access and perform administrative tasks using their new account.

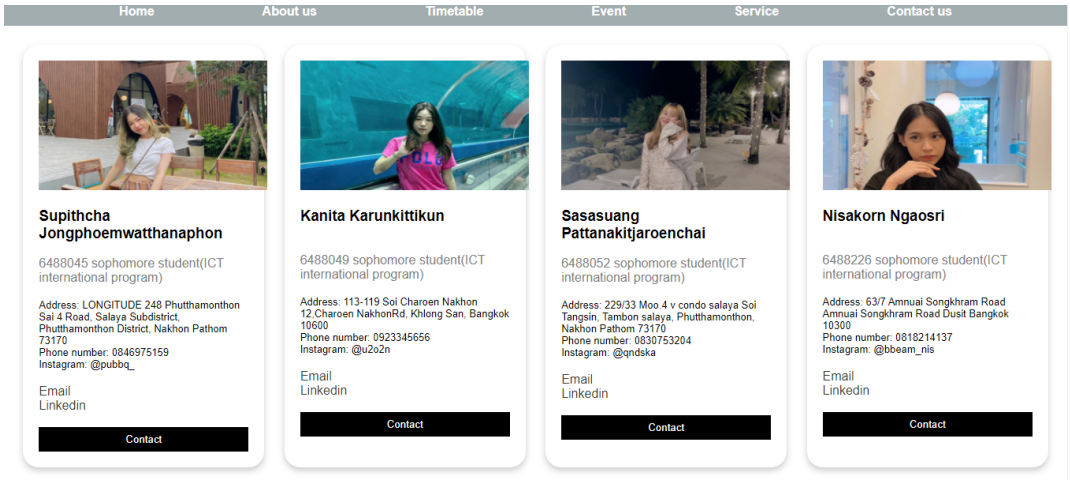


The purpose of the update account form is to update the information of an existing administrator by their admin ID. The data that can be updated include the administrator's name, username, email address, phone number, role, and address. After the user completes the information, the user can click the update button. It will update the new user information and perform administrative tasks using their new information.



If the administrator's information has been updated, a notification will appear as shown in the picture. And then, it will redirect to the Usermanage page and show the updated administrator's information.

About Us page:



URL: http://localhost:3030/aboutus

This page displays the creator's profile image, and personal details, including their name, contact information such as an address, phone number, email, and social media such as Instagram and LinkedIn. Additionally, it has "Email' and "LinkIn" buttons for communicating with the creator directly. This page also features a navigation bar that allows users to navigate to other pages without returning to the homepage.

Web application code

In addition to HTML and CSS files, we also use JavaScript to connect our web applications, which contain route.js and authen.js files.

route_ui .js file:

```
const express = require('express');
const path = require('path')
const port = 3030;
const app = express();
const router = express.Router();
app.use('/', express.static(path.join(__dirname, '/html')));
app.use('/js', express.static(path.join(__dirname, '/js')));
app.use(router);


router.get('/', (req, res) => {
    res.sendFile(path.join(`${__dirname}/html/Homepage.html`))
});
router.get('/aboutus', (req, res) => {
    res.sendFile(path.join(`${__dirname}/html/aboutus.html`))
    console.log('Visited on port 3000, About us page')
});
router.get('/login', (req, res) => {
    res.sendFile(path.join(`${__dirname}/html/login.html`))
});
//router post login
router.get('/signup', (req, res) => {
    res.sendFile(path.join(`${__dirname}/html/signup.html`))
});
//router post signup
router.get('/productmanage', (req, res) => {
    res.sendFile(path.join(`${__dirname}/html/productmanage.html`))
});
//router post product manage
router.get('/usermanage', (req, res) => {
    res.sendFile(path.join(`${__dirname}/html/usermanage.html`))
});
router.get('/welcome', (req, res) => {
    res.sendFile(path.join(`${__dirname}/html/welcome.html`))
});
router.get('/search', (req, res) => {
    res.sendFile(__dirname + '/html/search.html');
});
router.get('/detail1', (req, res) => {
    res.sendFile(path.join(`${__dirname}/html/detail1.html`))
});

router.use((req, res, next) => {
    res.status(404)
    console.log("404: Invalid accessed")
})
app.listen(port, function () {
    console.log(`Server listening on port: ${port}`)
    console.log(`Server for ui_website `)
});
```

Route_ui .js is a Node.js web application that uses the Express framework to serve static HTML, CSS, and JavaScript files to handle HTTP requests. This file has a purpose to serve a static website with different pages for different URLs. It demonstrates the use of routing in Express and the serving of static files.

This code starts with the express and path modules and defines the port variable to specify the port number on which the server should listen, so this server listens on port 3030.

The “router.get()” method is used to define a router for different URLs, and each route sends a corresponding HTML file using the res.sendFile( ) method. Therefore, if users entered an invalid URL, the server will respond with a 404 error message.

Example:

authen.js file:

This file includes functions for login, authentication, adding, updating, and deleting tickets and administrator information, as well as modifying the ticket list that will show the result.

```
function login(){
    let username = document.getElementById("username").value;
    let password = document.getElementById("password").value;
    let data = {
        "username": username,
        "password": password
    };
    fetch("http://localhost:3000/db_login", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
            },
            body: JSON.stringify(data)
        })
        .then(response => response.json())
        .then(data => {
            if (data.status == "success") {
                localStorage.setItem("token", data.token);
                window.location.href = "/welcome";
            } else {
                alert("Invalid username or password");
            }
        })
        .catch((error) => {
            console.error('Error:', error);
    });
}
```

Login function

```
function authen() {
    const token = localStorage.getItem('token');
    fetch('http://localhost:3000/db_authen', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Authorization': 'Bearer ' + token
        },
    })
        .then(response => response.json())
        .then(data_authen => {
            if (data_authen.message == 'Token is valid') {
                //pass
            } else {
                localStorage.removeItem('token');
                alert('Please login first!')
                window.location.href = '/login';
            }
            console.log('Success:', data_authen);
        })
        .catch((error) => {
            console.error('Error:', error);
        });
}
```

Authen function

- The login function sends a POST request to "http://localhost:3000/db_login" with the entered username and password in JSON format. If the response status is "success," the server sends back a token that is stored in localStorage, and the user is redirected to the "/welcome" page on the web application. Otherwise, an alert is displayed with the message "Invalid username or password."
- The Authen function sends a POST request to "http://localhost:3000/db_authen" with the token stored in localStorage in the Authorization header. If the response message is "Token is valid," the function returns. Otherwise, the token is removed from localStorage, an alert is displayed with the message "Please login first!", and the user is redirected to the "/login" page on the web application.

```
function addticket(){
    let startdes = document.getElementById("startdes").value;
    let endtdes = document.getElementById("endtdes").value;
    let tickettype = document.getElementById("tickettype").value;
    let trainnum = document.getElementById("trainnum").value;
    let date = document.getElementById("date").value;
    let data = {
        "departure_station": startdes,
        "terminal_station": endtdes,
        "ticket_type": tickettype,
        "train_number": trainnum,
        "available_date": date
    };
    fetch("http://localhost:3000/addtickets", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
            },
            body: JSON.stringify(data)
        })
        .then(response => response.json())
        .then(data => {
            if (data.status == "success") {
                alert("Add ticket success");
                //reload page
                window.location.reload();
            } else {
                alert("Error!! Please try again");
            }
        })
        .catch((error) => {
            console.error('Error:', error);
            alert("Error!! Please try again");
    });
}
```

addticket function

```
function deleteticket(ticket_id){
    let data = {
        "ticket_id": ticket_id
    };

    fetch("http://localhost:3000/deletetickets", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
            },
            body: JSON.stringify(data)
        })
        .then(response => response.json())
        .then(data => {
            if (data.status == "success") {
                alert("Delete ticket success");
                //reload page
                window.location.reload();
            } else {
                alert("Error!! Please try again");
            }
        })
        .catch((error) => {
            console.error('Error:', error);
            alert("Error!! Please try again");
    });
}
```

deleteticket function

- The addticket function sends a POST request to "http://localhost:3000/addtickets" with the entered ticket details from the product management page including departure station, terminal station, ticket type, train number, and available date in JSON format. If the response status is "success," an alert is displayed with the message "Add ticket success," and the page is reloaded. Otherwise, an alert is displayed with the message "Error!! Please try again."
- The deleteticket function sends a POST request to "http://localhost:3000/deletetickets" with the ticket ID to be deleted in JSON format. If the response status is "success," an alert is displayed with the message "Delete ticket success," and the page is reloaded. Otherwise, an alert is displayed with the message "Error!! Please try again."

```
// update ticket
function updateticket(){
    let tiketid = document.getElementById("tiketid").value;
    let departure_station = document.getElementById("startdes").value;
    let terminal_station = document.getElementById("endtdes").value;
    let ticket_type = document.getElementById("tickettype").value;
    let train_number = document.getElementById("trainnum").value;
    let available_date = document.getElementById("date").value;

    fetch("http://localhost:3000/updatetickets", {
        method: "PUT",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            new_tickets : {
            id: tiketid,
            departure_station: departure_station,
            terminal_station: terminal_station,
            ticket_type: ticket_type,
            train_number: train_number,
            available_date: available_date
            }
        })
    })
    .then(response => response.json())
    .then(res => {
        if (res.status == "success"){
            alert('Update ticket success');
            // window.location.href = "/usermanage";
            location.reload();
        }
        else {
            alert("Error update ticket")
        }
    })
    .catch((error) => {
        console.error('Error:', error);
        alert("Error to update!! Please try again");
    });
}
```

updateticket function

```
function modifyboxlist(){
    let data = {
        "departure_station": "",
        "terminal_station": "",
        "available_date": ""
    };
    fetch("http://localhost:3000/gettickets", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify(data)
    })
    .then(response => response.json())
    .then(res => {
        console.log(res.data);
        let modifyticketbox = document.getElementById("modifyticketbox");
        modifyticketbox = "<br><br>";
        for (let i = 0; i < res.data.length; i++) {
            //create modify ticket box
            modifyticketbox += `
                <div class="modifyticketbox"
                    <p>Ticket ${i+1} <br> ${res.data[i].departure_station} >> ${res.data[i].terminal_station} </p><br>
                    <p>Type: ${res.data[i].ticket_type}</p>
                    <p>Train number: ${res.data[i].train_number}</p>
                    <p>Available Date: ${res.data[i].available_date}</p>
                    <div class="right">
                        <button class="editt"><a href="#"><p>Edit</p></a></i>
                        </button>
                        <button class="deletee"><a href="#" onclick=deleteticket(${res.data[i].id})><p>Delete</p></a></i>
                        </button>
                    </div>
                </div>
            `
        }
        document.getElementById("modifyticketbox").innerHTML = modifyticketbox;
    })
    .catch((error) => {
        console.error('Error:', error);
    });
}
```

modifyboxlist function

- The updateticket function is responsible for updating a ticket in a database. It sends a PUT request to the server at "http://localhost:3000/updatetickets". The function first gets the values of different elements on the webpage using their id attributes and stores them

in variables. It then creates a JSON object with these variables and sends it as the body of the request. After receiving the response, the function checks if the status property of the response is "success". If it is, it displays an alert saying "Update ticket success" and reloads the page. Otherwise, it displays an alert saying "Error update ticket".

- The modifyboxlist function is responsible for retrieving a list of tickets from the server and displaying them on the product manage page. The function sends a POST request to the server at "http://localhost:3000/gettickets" with an empty JSON object as the body. After receiving the response, the function retrieves the data property from the response object and loops through each ticket. For each ticket, it creates an HTML element with its details and appends it to a variable called modifyticketbox. Finally, it sets the inner HTML of an element with id attribute modifyticketbox to this variable, thereby displaying the list of tickets on the product manage page.
- [Appendix A] The search function performs a search operation by sending a POST request to "http://localhost:3000/gettickets" with a JSON payload containing the search criteria (departure_station, terminal_station, and available_date). It then retrieves the response data and populates the search result section of the search page with the relevant data.
- [Appendix B] The getadmin function retrieves a list of admins by sending a POST request to "http://localhost:3000/getadmin" with a JSON payload containing the search criteria (adminID, admin_username, and admin_fullname). It then retrieves the response data and populates the search user result section of the usermanage page with the relevant data. The search results are displayed as a list of users with their profile picture, email address, and full name. Each Administrator has "Edit" and "Delete" buttons, if they press the edit button the website will be sent to the signin page for editing or adding information, but if pressing the delete button that specific administrator information will be deleted.

```
function deleteadmin(adminID){
    fetch("http://localhost:3000/deleteadmin", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            adminID: adminID
        })
    })
    .then(response => response.json())
    .then(res => {
        console.log(res.data);
        if (res.status == "success"){
            //reload page
            location.reload();
        }
    })
    .catch((error) => {
        console.error('Error delete admin:', error);
    });
}
```

deleteadmin function

```
function updateadmin(admin_IDup){
    let admin_fullname = document.getElementById("admin_fullnameup").value;
    let admin_username = document.getElementById("admin_usernameup").value;
    let admin_email = document.getElementById("admin_emailup").value;
    let admin_phonenum = document.getElementById("admin_phonenumup").value;
    let admin_role = document.getElementById("admin_roleup").value;
    let admin_address = document.getElementById("admin_addressup").value;
    let admin_password = document.getElementById("admin_passwordup").value;
    let admin_confpassword = document.getElementById("admin_confpasswordup").value;
    let adminID = document.getElementById("admin_IDup").value;

    fetch("http://localhost:3000/updateadmin", {
        method: "PUT",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            adminregister : {
            adminID: adminID,
            admin_fullname: admin_fullname,
            admin_username: admin_username,
            admin_email: admin_email,
            admin_phonenum: admin_phonenum,
            admin_role: admin_role,
            admin_address: admin_address,
            admin_password: admin_password,
            admin_confpassword: admin_confpassword
            }
        })
    })
    .then(response => response.json())
    .then(res => {
        if (res.status == "success"){
            alert('Update admin success');
            // window.location.href = "/usermanage";
            location.reload();
        }
        else {
            alert("Error update admin")
        }
    })
    .catch((error) => {
        console.error('Error:', error);
        alert("Error to update!! Please try again");
    });
}
```

updateadmin function

- The deleteadmin function takes an adminID parameter, sends a POST request to the server at "http://localhost:3000/deleteadmin" with the Content-Type header set to application/json, and the request body containing a JSON object with the adminID property set to the provided adminID parameter. If the response's status property is "success", it reloads the current page, otherwise it logs an error message to the console.
- The updateadmin function takes an admin_IDup parameter and updates an existing administrator's information based on an adminID. It sends a PUT request to the server at "http://localhost:3000/updateadmin" with the Content-Type header set to application/json, and the request body containing a JSON object with properties for the new administrator information, including adminID, admin_fullname, admin_username, admin_email, admin_phonenum, admin_role, admin_address, admin_password, and

admin_confpassword. If the response's status property is "success", it shows an alert with the message "Update admin success" and reloads the current page, otherwise it shows an alert with the message "Error update admin".

- adminID that used as a parameter gets from the search function at this line of code:

```
<div class="right">
<button class="edit"><a onclick="updateadmin(${res.data[i].adminID})" href="/signup" >Edit</a></i></button>
<button class="delete" onclick="deleteadmin(${res.data[i].adminID})"><a href="#" >Delete</a></i> </button>
</div>
```

- [Appendix C] The addadmin function used for adds a new administrator to the system. It sends a POST request to the server at "http://localhost:3000/add_admin" with the Content-Type header set to application/json, and the request body containing a JSON object with properties for the new administrator information, including admin_fullname, admin_username, admin_email, admin_phonenum, admin_role, admin_address, admin_password, and admin_confpassword. If the admin_password and admin_confpassword fields do not match or if the admin_password field has a length greater than 8, it shows an alert with a relevant error message. Otherwise, if the response's status property is "success", it shows an alert with the message "Add admin success" and navigates to the "/usermanage" page, otherwise it shows an alert with the message "Error".

Details of web service and code

Source code directory files for web service:

- railwayticket.sql
- task2_db.js

railwayticket.sql file:

railwayticket.sql is a file that contains the SQL (Structured Query Language) commands to create and manage three databases related to ticket information and administrator login for a web application.

These databases include:

- Administrator Login Information Database - Table `adminlogin` stores information that needs to be checked when administrators attempt to log in, which are adminID, admin_username, and admin_password. This information is used to authenticate and authorize administrators when they log in to the web application.

```
CREATE TABLE `adminlogin` (
  `adminID` int(50) NOT NULL,
  `admin_username` varchar(50) NOT NULL,
  `admin_password` varchar(8) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Dumping data for table `adminlogin`
--

INSERT INTO `adminlogin` (`adminID`, `admin_username`, `admin_password`) VALUES
(1, 'foxzfoxzfoxz', '08041999'),
(2, 'kageyama', '22101996'),
(3, 'toruoikawa', '20071994'),
(4, 'garnets', '07072004'),
(5, 'tsukishima', '27091996'),
(6, 'tetsurokuroo', '17111994'),
(7, 'hinata', '21061996'),
(8, 'nishinoya', '10101995');
```

- Administration Information Database - Table `adminregister` contains information related to the administrators of the web application, such as their ID, names, email addresses, role, phone numbers, etc. This information is used by the web application to search, view, update, and delete administrator information.

```sql
--
-- Table structure for table `adminregister`
--

CREATE TABLE `adminregister` (
  `adminID` int(11) NOT NULL,
  `admin_fullname` varchar(50) NOT NULL,
  `admin_username` varchar(50) NOT NULL,
  `admin_email` varchar(100) NOT NULL,
  `admin_phonenum` varchar(50) NOT NULL,
  `admin_role` varchar(50) NOT NULL,
  `admin_address` varchar(300) NOT NULL,
  `admin_password` varchar(8) NOT NULL,
  `admin_confpassword` varchar(8) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Dumping data for table `adminregister`
--

INSERT INTO `adminregister` (`adminID`, `admin_fullname`, `admin_username`, `admin_email`, `admin_phonenum`, `admin_role`, `admin_address`, `admin_password`, `admin_confpassword`)
VALUES
(99, 'Sasasuang Pattanakitjaroenchai', 'Sasasuang', 'Sasasuang.pat@student.mahidol.edu', '0830753204', 'manager', '229/33', '12345678', '12345678'),
(100, 'Kanita Karunkittikun', 'Kanita', 'Kanita.kar@student.mahidol.edu', '0923345656', 'creator', '229/23', '12343478', '12343478'),
(101, 'Supithcha Jongphoemwatthanaphon', 'Supithcha', 'Supithcha.jon@student.mahidol.edu', '0846975159', 'manager', '229/11', '11113478', '11113478'),
(102, 'Nisakorn Ngaosri', 'Nisakorn', 'Nisakorn.nga@student.mahidol.edu', '0818214137', 'creator', '229/17', '11143478', '11143478'),
(111, 'Manee Meethong', 'Manee', 'Manee.mee@student.mahidol.edu', '0800751590', 'admin', '229/11', '11113888', '11113478');
```

- Ticket Information Database - Table `new_tickets` contains information related to the tickets, such as the ticket ID, departure station, terminal station, ticket type, train number, and available date for that ticket. This information is used by the web application to search, view, update, and delete ticket information.

```sql
--
-- Table structure for table `new_tickets`
--

CREATE TABLE `new_tickets` (
  `id` int(11) NOT NULL,
  `departure_station` varchar(255) NOT NULL,
  `terminal_station` varchar(255) NOT NULL,
  `ticket_type` varchar(255) NOT NULL,
  `train_number` varchar(255) NOT NULL,
  `available_date` varchar(255) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Dumping data for table `new_tickets`
--

INSERT INTO `new_tickets` (`id`, `departure_station`, `terminal_station`, `ticket_type`, `train_number`, `available_date`) VALUES
(2, 'Chiang Mai', 'Kra bi', 'test', '5', '2014-04-04'),
(5, 'Chiang Mai', 'Chang Rai', 'asd', '555', '2023-04-15'),
(6, 'Kra Bi', 'Bangkok', 'SDA', '4', '2023-04-19'),
(7, 'Kra Bi', 'Chiang Mai', 'KAS', '6', '2023-04-23'),
(8, 'Kra Bi', 'Bangkok', 'SDA', '4', '2023-04-22'),
(9, 'Kra Bi', 'Bangkok', 'SAD', '6', '2023-04-22'),
(10, 'Bangkok', 'Kra Bi', 'FAS', '6', '2023-04-23');
```

Task2_db.js file:

Task2_db.js is a file that plays a crucial role in verifying the information before displaying it on the website. It retrieves the required data from the website and utilizes the router command to connect with the database and validate the information before displaying it.
This file includes:

```
// 2.1 Authentication Service
router.post('/db_login', function (req, res) {
    let username = req.body.username;
    dbConn.query('SELECT * FROM adminlogin WHERE admin_username = ? AND admin_password = ?', [req.body.username, req.body.password],
    function (error, results, fields) {
        if (error){
            res.json({'status': 'error', 'message': error})
        }else if(results.length == 0){
            res.json({'status': 'error', 'message': 'Invalid username or password'})
        }else{
            //create token
            let tokens = jwt.sign({username}, 'secretkey', {expiresIn: '1h'});
            res.json({status: 'success', token: tokens, message: 'Login Success'})
        }
    });
});
```

- To authenticate the admin during login, a POST request must be sent to "http://localhost:3000/db_login" containing the username and password information. The server will then check if the information provided is valid, and if it is, a message "Login Success" will be displayed. However, if the information is invalid, an error message with "Invalid username and password" will be shown.

```
router.post('/db_authen', function (req, res) {
    try {
        let checktoken = req.headers.authorization.split(' ')[1];
        let decoded = jwt.verify(checktoken, 'secretkey');
        res.json({status: 'success', message: 'Token is valid'})
    }
    catch (err) {
        res.json({status: 'error', message: 'Token is invalid'})
    }
});
```

- To check the token, a POST request needs to be sent to "http://localhost:3000/db_authen". If the token is valid, the response will show "success" status and the message" Token is valid". However, if the token is invalid, a " Token is invalid" message will be displayed and show an "error" status.

```
// 2.2 Administrators Service
// SELECT admin info
router.post('/getadmin', function (req, res) {
    let admin_id = req.body.adminID;
    let admin_username = req.body.admin_username;
    let admin_fullname = req.body.admin_fullname;
    dbConn.query('SELECT * FROM adminregister', function (error, results, fields) {
        if (error) throw error;
        else{
            let query_json = [];
            let count = 0;
            for(let i = 0; i < results.length; i++){
                if (admin_id != "" && results[i].adminID != admin_id){
                    continue;
                }
                if (admin_username != "" && results[i].admin_username != admin_username){
                    continue;
                }
                if (admin_fullname != "" && results[i].admin_fullname != admin_fullname){
                    continue;
                }
                query_json[count] = results[i];
                console.log(results[i])
                count++;
            }
            return res.send({ status: 'success', data: query_json, message: 'Admin list.' });
        }

    });
});
```

- To search for admin's information, a POST request must be sent to "http://localhost:3000/getadmin". The server will then check the provided adminID, admin username, and admin full name against the information in the database. If the information matches, the response will show a status of "success" along with the admin's information. If there is no matching admin's information found in the database, an error status will be displayed.

```
// INSERT admin
router.post('/add_admin', function (req, res) {
    try{
        dbConn.query('INSERT INTO adminregister (admin_fullname, admin_username, admin_email, admin_phonenum, admin_role, admin_address, admin_password, admin_confpassword) VALUES (?,?,?
        [req.body.admin_fullname, req.body.admin_username, req.body.admin_email, req.body.admin_phonenum, req.body.admin_role, req.body.admin_address, req.body.admin_password, req.body.a
        function (error, results, fields) {
            if (error) throw error;
            return res.send({ status: 'success', data: results, message: 'Admin has been created successfully.' });
        });
    }
    catch(error){
        console.log(error);
    }
});
```

- To add a new admin's information, a POST request must be sent to "http://localhost:3000/add_admin". The server will then prompt the user to provide all the required details that match the existing information in the database. If the user provides all the necessary information, a message will be displayed indicating that the admin has

been created successfully. However, if the information provided is incomplete or incorrect, an error message will be displayed.

```
router.post('/deleteadmin', function (req, res) {
    dbConn.query('DELETE FROM adminregister WHERE adminID = ?', [req.body.adminID], function (error, results) {
        if (error) throw error;
        return res.send({ status: 'success', data: results.affectedRows, message: 'Admin has been deleted successfully.' });
    });
});
```

- To delete an admin's information, a POST request must be sent to "http://localhost:3000/deleteadmin" containing the admin ID. If the admin ID is present in the database, a message will be displayed indicating that the admin has been deleted successfully. However, if the admin ID is not found in the database, an error message will be displayed.

```
//UPDATE admin
router.put('/updateadmin', function (req, res) {
    let adminregister = req.body.adminregister;
    console.log("admin body:", adminregister)
    let adminID = req.body.adminregister.adminID;
    console.log("adminID", adminID)
    if (!adminID || !adminregister) {
        return res.status(400).send({ error: adminregister, message: 'Please provide admin information' });
    }
    dbConn.query("UPDATE adminregister SET ? WHERE adminID = ?", [adminregister, adminID], function (error, results) {
        if (error) throw error;
        return res.send({ status: 'success', data: results.affectedRows, message: 'Admin has been updated successfully.' })
    });
    console.log("finish task2_bd > updateadmin");
});
```

- To update an admin's information, a PUT request must be sent to "http://localhost:3000/updateadmin" containing all the admin information. If the user provides all the required information correctly, a message will be displayed indicating that the admin has been updated successfully. However, if the information provided is incomplete or incorrect, a message "Please provide admin information" will be displayed.

```
//SELECT> search ticket
router.post('/gettickets', function (req, res) {
    let departure_station = req.body.departure_station;
    let terminal_station = req.body.terminal_station;
    let available_date = req.body.available_date;
    dbConn.query('SELECT * FROM new_tickets', function (error, results, fields) {
        if (error)
            throw error;
        else{
            //console.log(results)
            //create empty json
            let query_json = [];
            let count = 0;
            for (let i = 0; i < results.length; i++) {
                if (departure_station != "" && departure_station != results[i].departure_station) {
                    continue;
                }
                if (terminal_station != "" && terminal_station != results[i].terminal_station) {
                    continue;
                }
                if (available_date != "" && available_date != results[i].available_date) {
                    continue;
                }
                query_json[count] = results[i];
                count++;
            }
            return res.send({ status: 'success', data: query_json, message: 'Tickets list.' });
        }
    });
});
```

- To search for information about a ticket, a POST request should be sent to "http://localhost:3000/gettickets". The server will then check if the provided departure station, terminal station, and available date match any information in the database. If a match is found, the server will respond with a status of "success" and display the ticket information. If there is no matching ticket information found in the database, an error status will be displayed.

```
// UPDATE ticket
router.put('/updatetickets', function (req, res) {
    let id = req.body.new_tickets.id;
    console.log("ticketid:", id)
    let new_tickets = req.body.new_tickets;
    console.log("ticket body:",new_tickets)
    if (!id|| !new_tickets) {
        return res.status(400).send({ error: new_tickets, message: 'Please provide ticket information' });
    }
    dbConn.query("UPDATE new_tickets SET ? WHERE id = ?", [new_tickets, id], function (error, results) {
        if (error) throw error;
        return res.send({ status: 'success', error: false, data: results.affectedRows, message: 'Tickets has been updated successfully.' })
    });
});
```

- To update a ticket's information, the user needs to send a PUT request to "http://localhost:3000/updatetickets" with all the necessary ticket information. If the user provides complete and accurate information, a message will be displayed indicating that the ticket has been updated successfully. However, if the information provided is incomplete or incorrect, the message "Please provide ticket information" will be displayed instead.

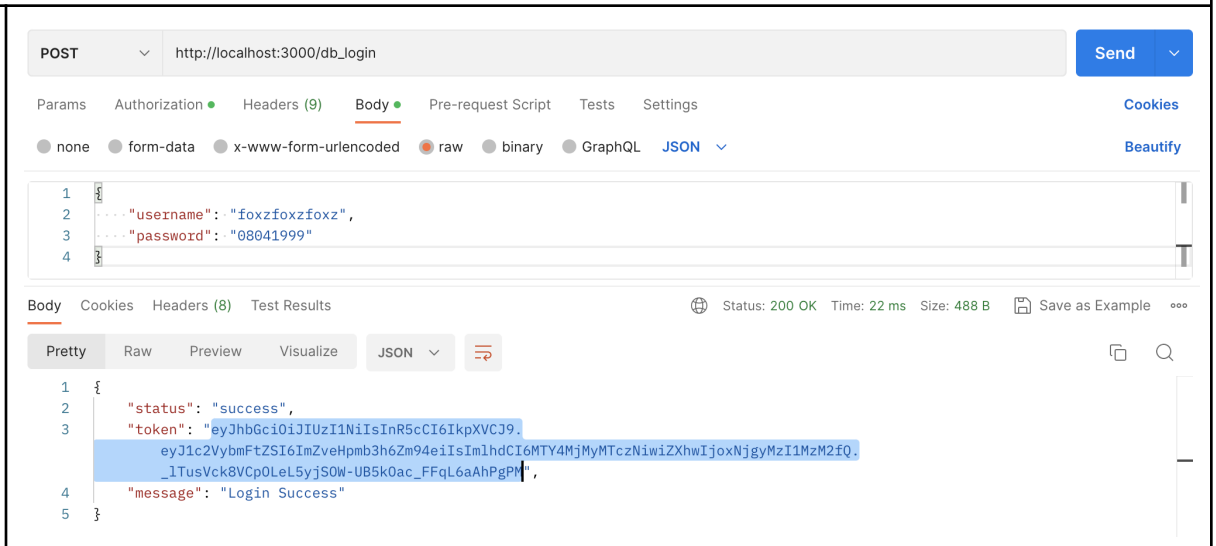
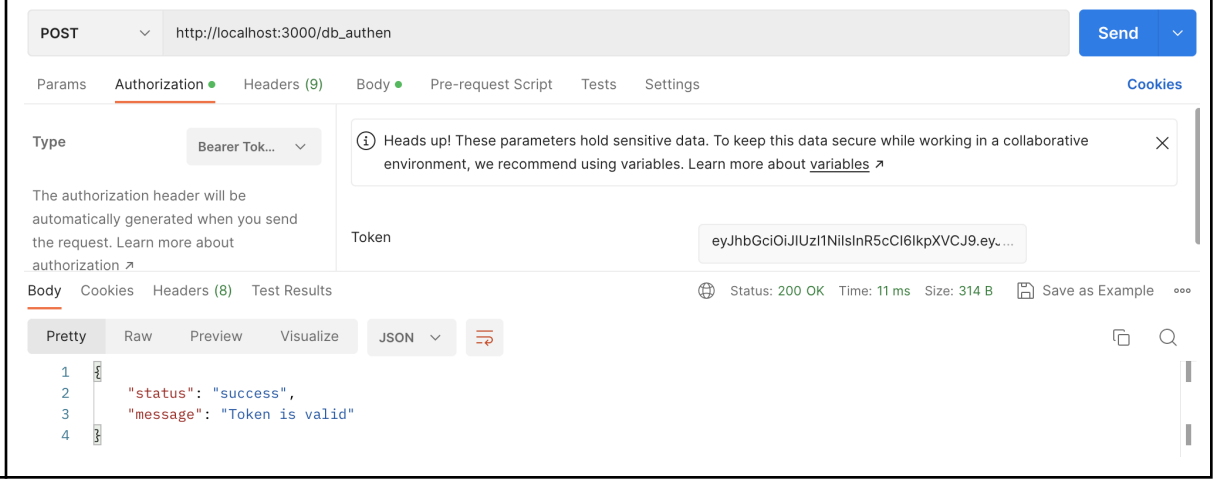
```
//INSERT new ticket
router.post('/addtickets', function (req, res) {
    let departure_station = req.body.departure_station;
    let terminal_station = req.body.terminal_station;
    let ticket_type = req.body.ticket_type;
    let train_number = req.body.train_number;
    let available_date = req.body.available_date;
    if (!departure_station || !terminal_station || !ticket_type || !train_number || !available_date) {
        return res.status(400).send({ status: 'infomation is have null', message: 'Please provide all required information' });
    }
    dbConn.query("INSERT INTO new_tickets (departure_station, terminal_station, ticket_type, train_number, available_date) VALUES (?, ?, ?,
    function (error, results) {
        if(error){
            res.json({status: 'error' , message: error});
        }else{
            res.json({status: 'success' , message: 'Ticket added successfully'});
        }
    });
});
```

- To add a new ticket's information, a POST request should be sent to "http://localhost:3000/addtickets". The server will then ask the user to provide all the required details that match the existing information in the database. If the user provides all the necessary information correctly, a message indicating that the ticket has been added successfully will be displayed. However, if the information provided is incomplete or incorrect, the message "Please provide all required information" will be displayed

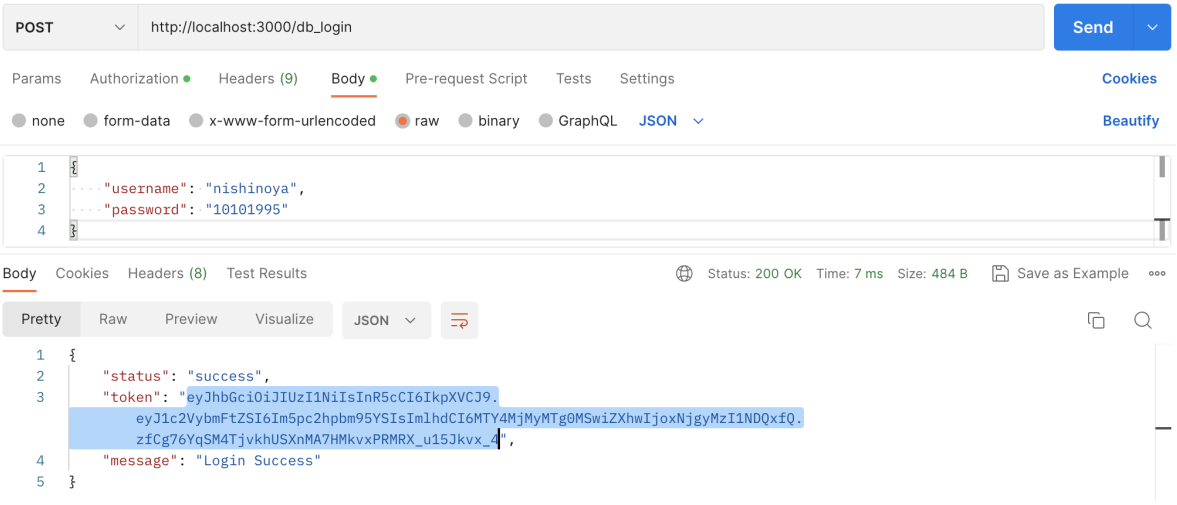
```
router.post('/deletetickets', function (req, res) {
    let ticket_id = req.body.ticket_id;
    if (!ticket_id) {
        return res.status(400).send({ error: true, message: 'Please provide ticket_id' });
    }
    dbConn.query('DELETE FROM new_tickets WHERE id = ?', [ticket_id], function (error, results) {
        if (error){
            res.json({status: 'error' , message: error});
        }
        else{
            res.json({status: 'success' , message: 'Ticket has been deleted successfully.' });
        }
    });
})
```

- To delete a ticket's information, a POST request should be sent to "http://localhost:3000/deletetickets" with the ticket ID. If the provided ticket ID exists in the database, a message will be displayed indicating that the ticket has been successfully deleted. However, if the provided ticket ID is not found in the database, An error message will be displayed.

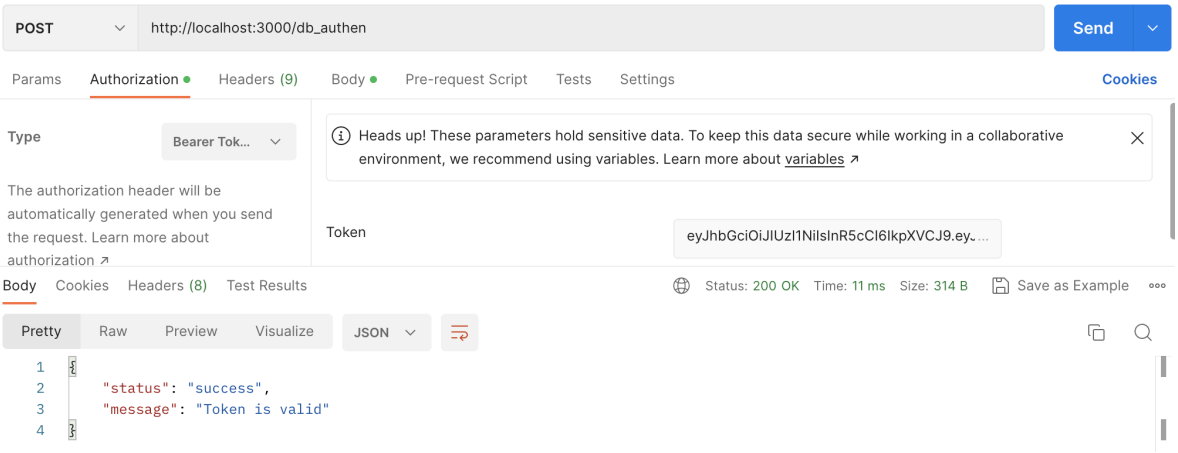
Testing results of web services via Postman

<table>
<tr><th colspan="2">Authentication Service</th></tr>
<tr><td>Login</td><td> 
</td></tr>
<tr><td>Test case1</td><td> 
</td></tr>
</table>

Login



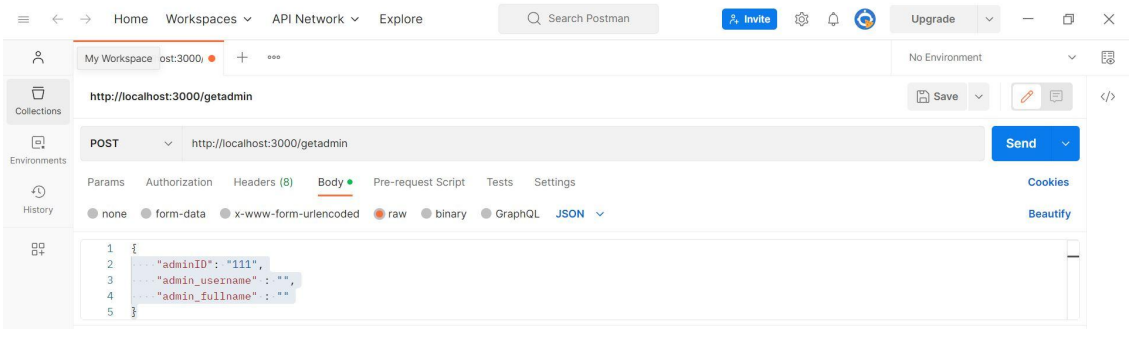
Test case2

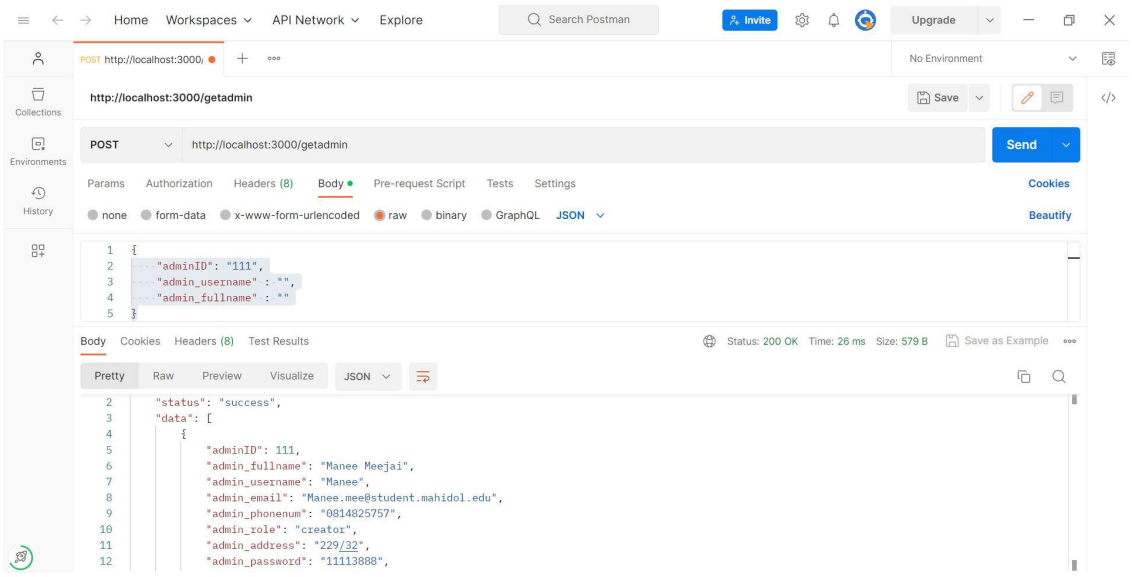
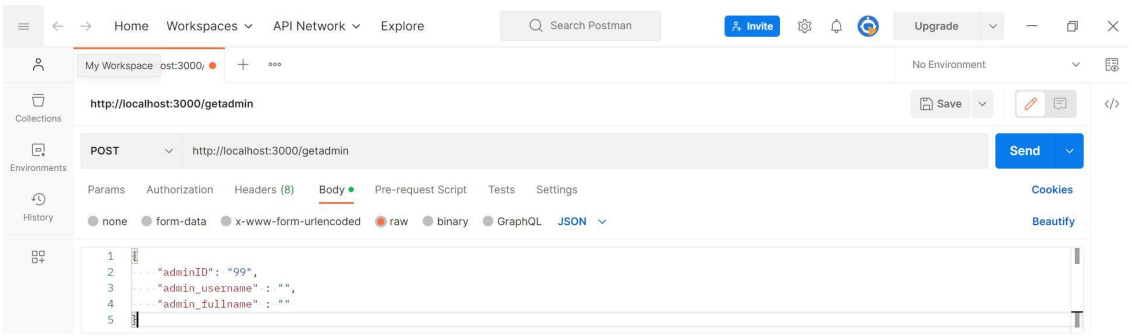
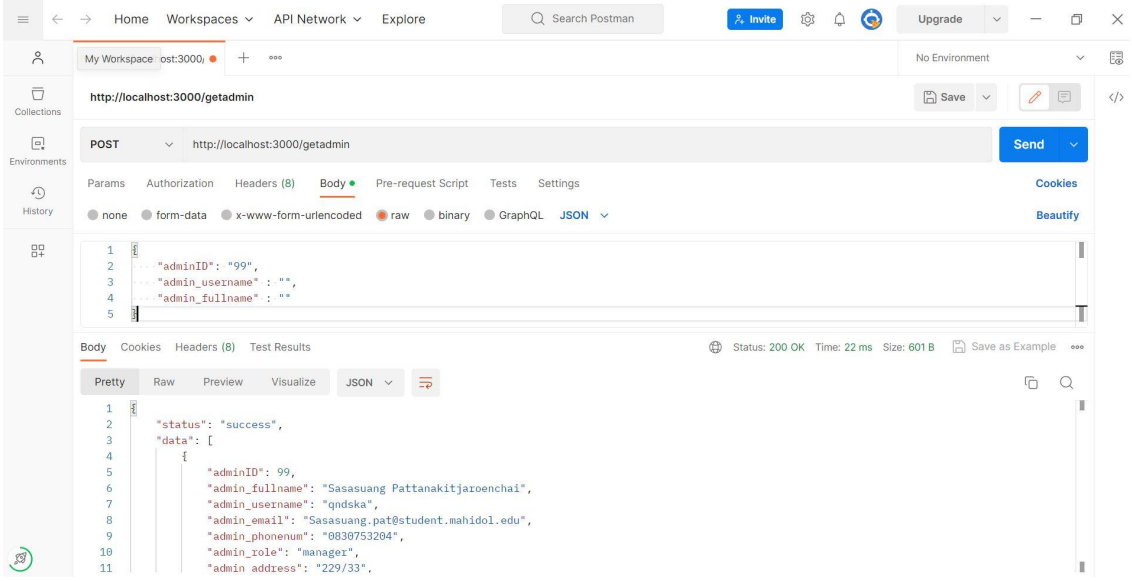


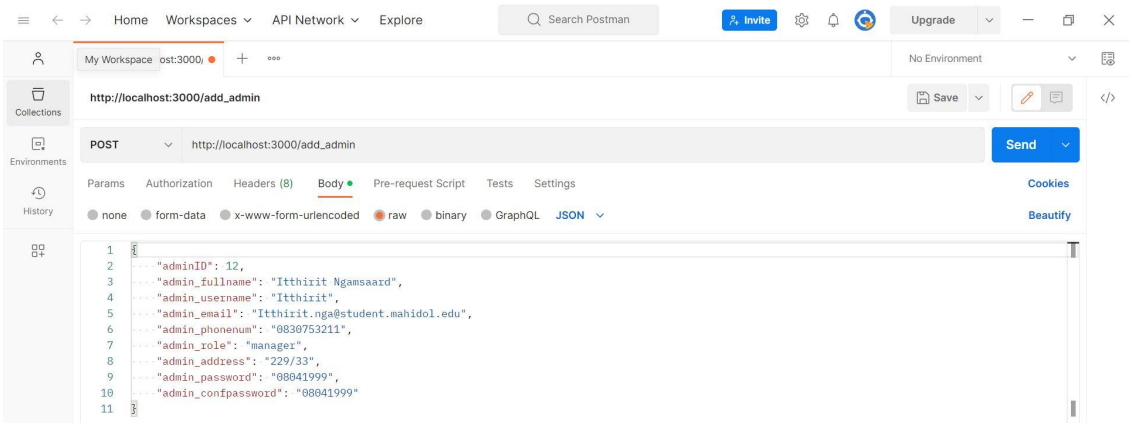
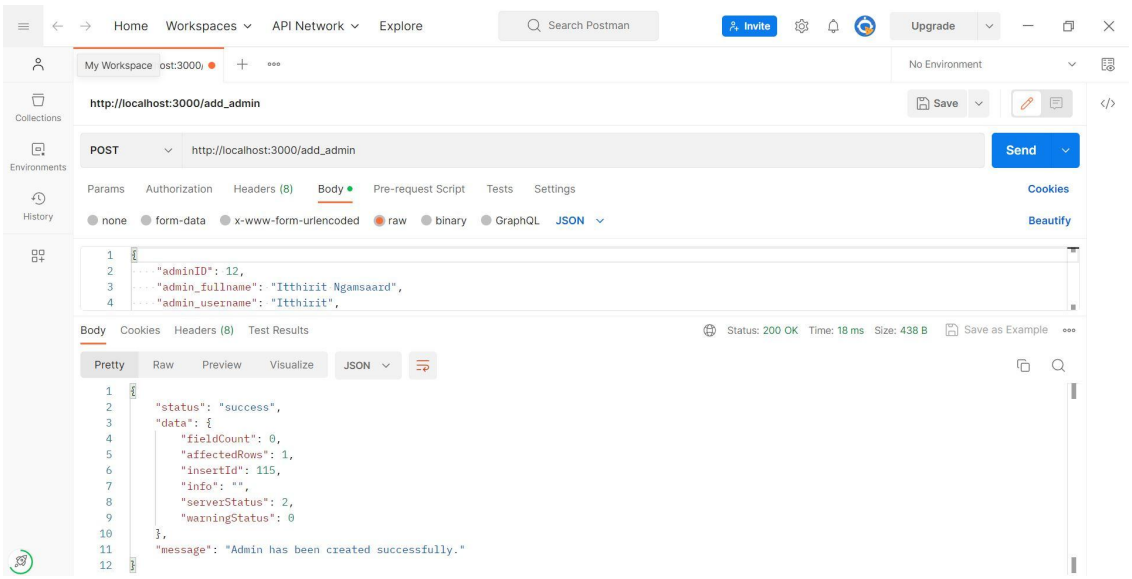
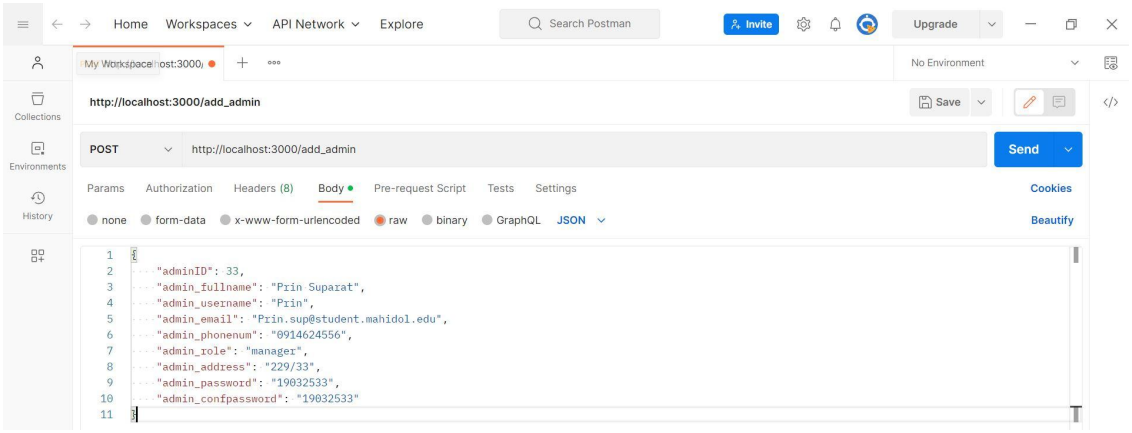
Administrator Service

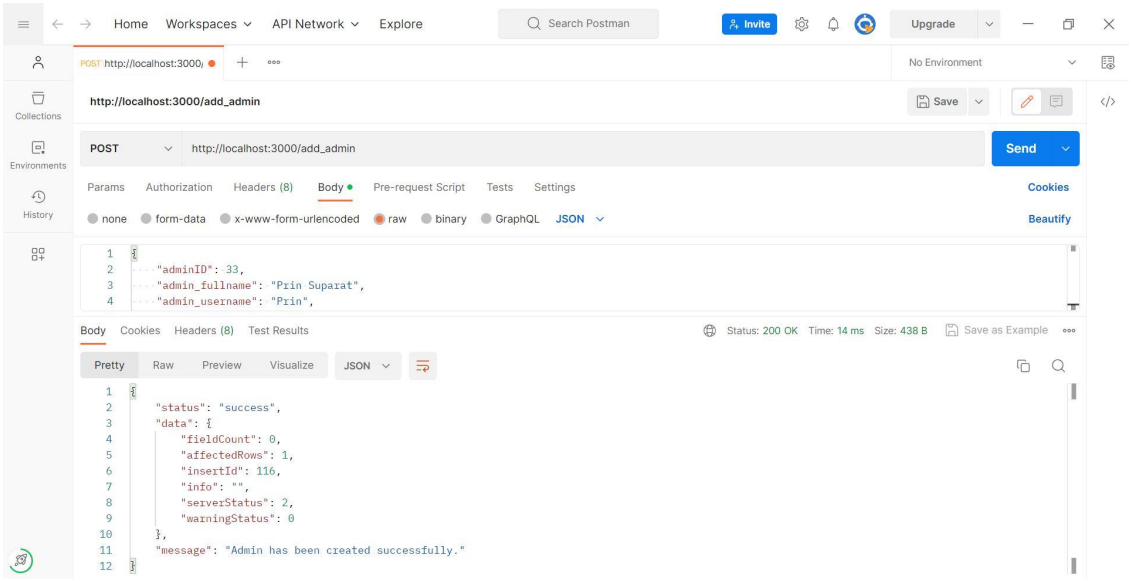
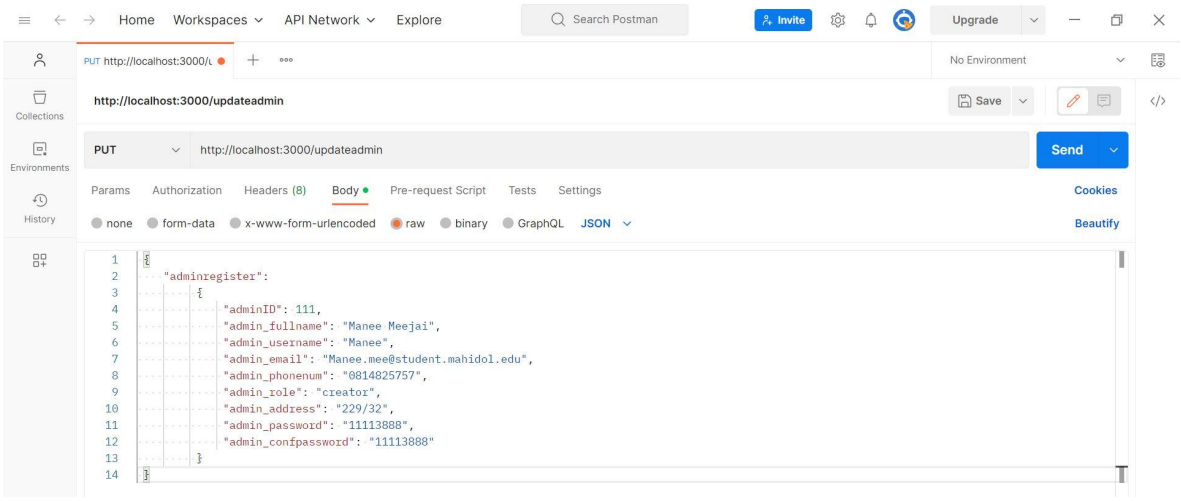
Administrator Search/View

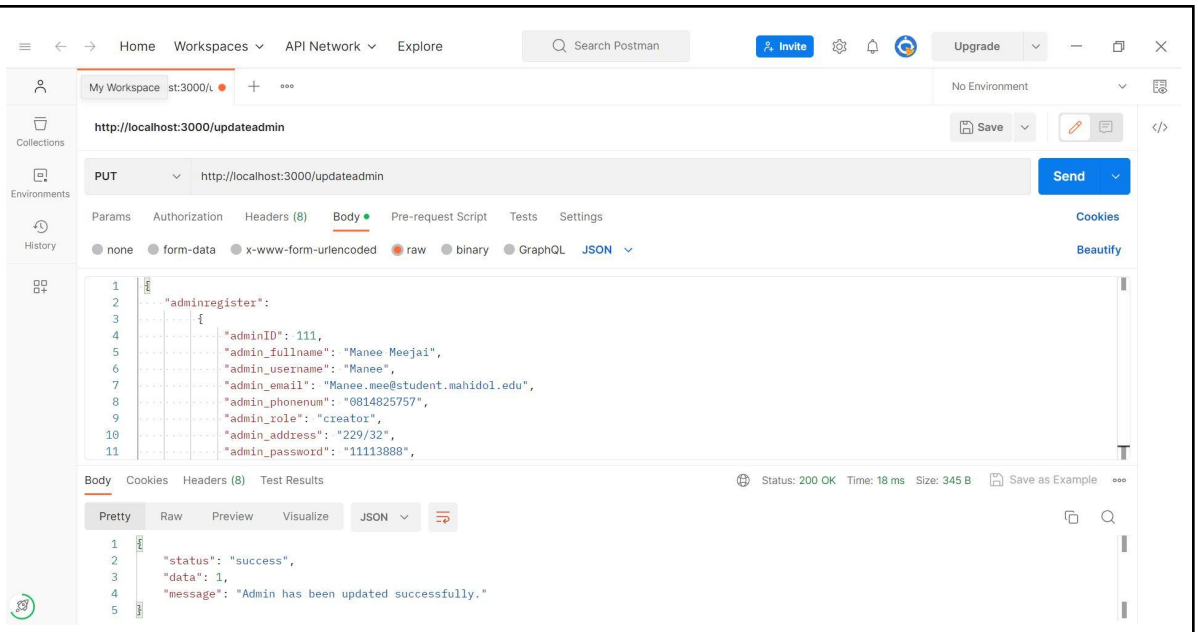
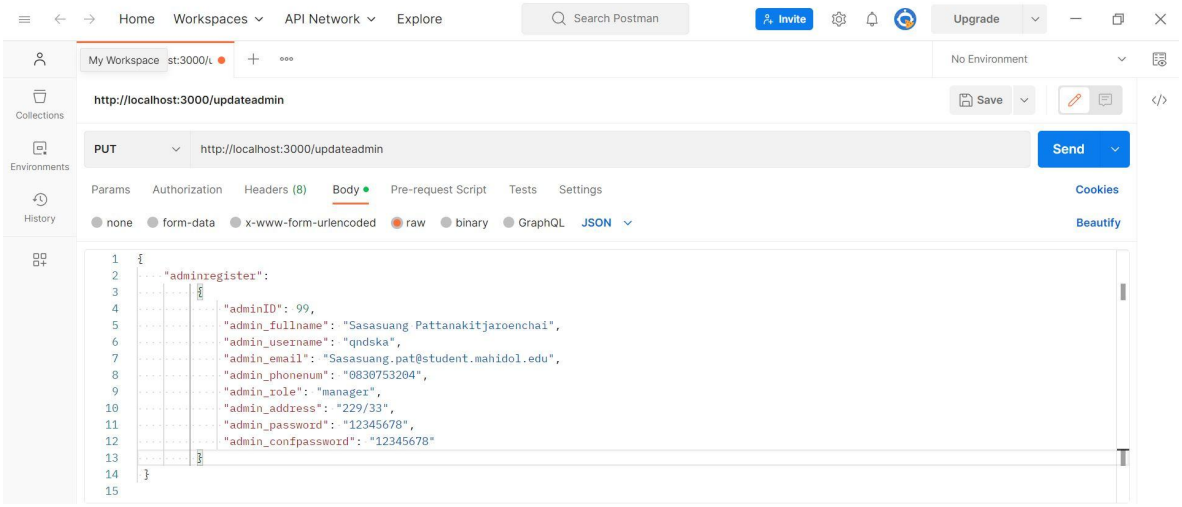
Getadmin



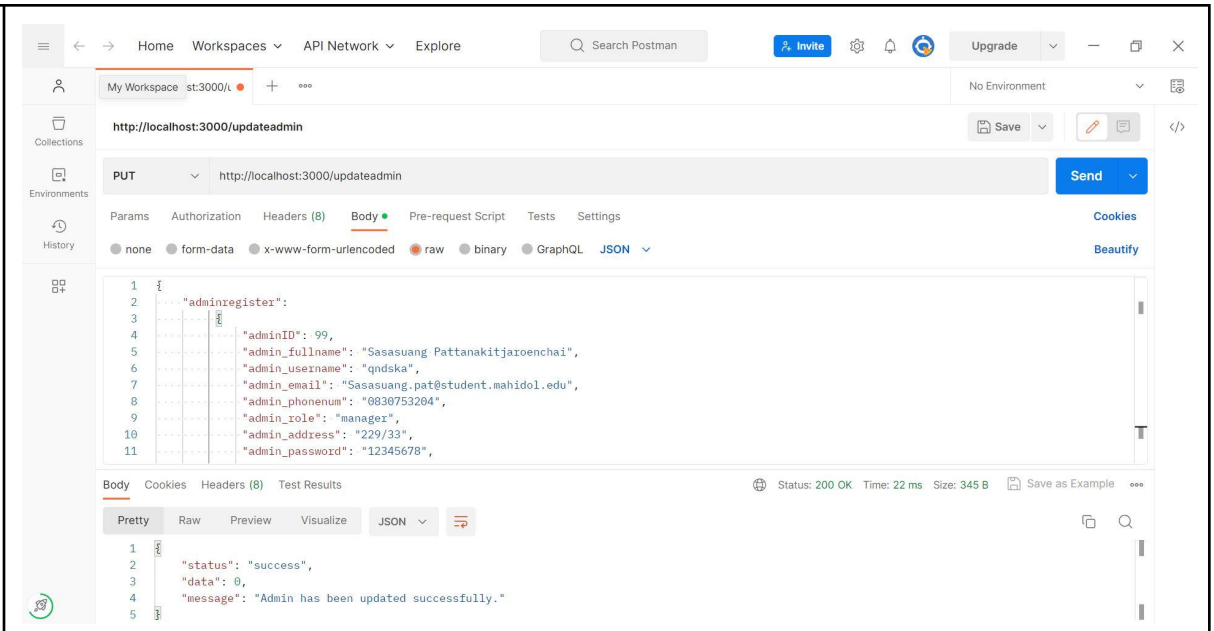
Test case1	
Getadmin	
Test case2	
Administrator Insert	

Add_admin	
Test case1	
Add_admin	

Test case2	
Administrator Update	
updateadmin	

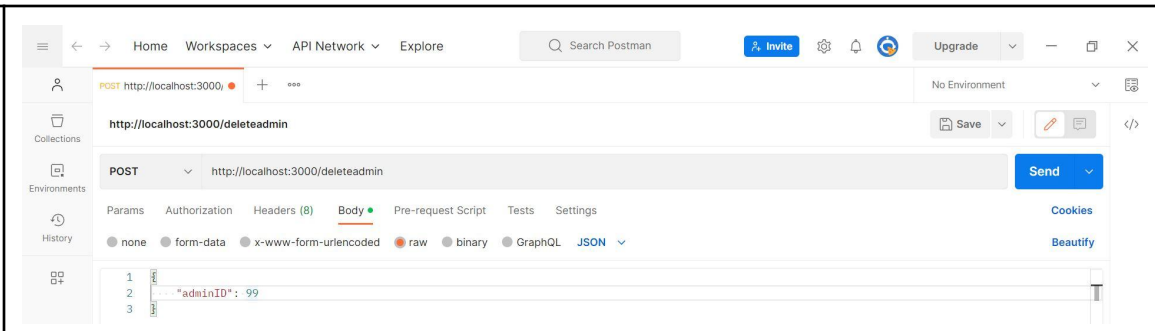
Test case1	
updateadmin	

Test case2

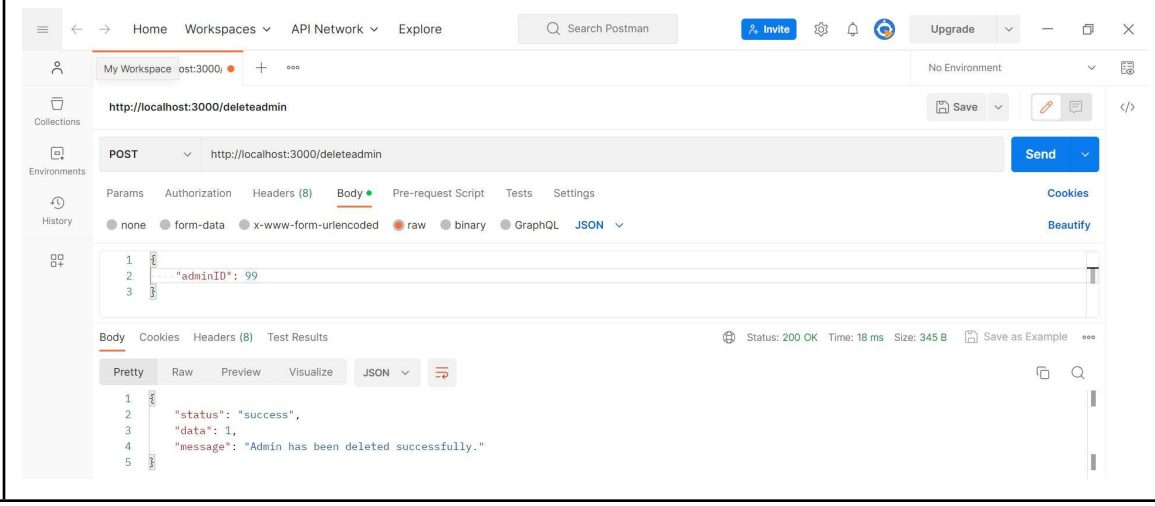


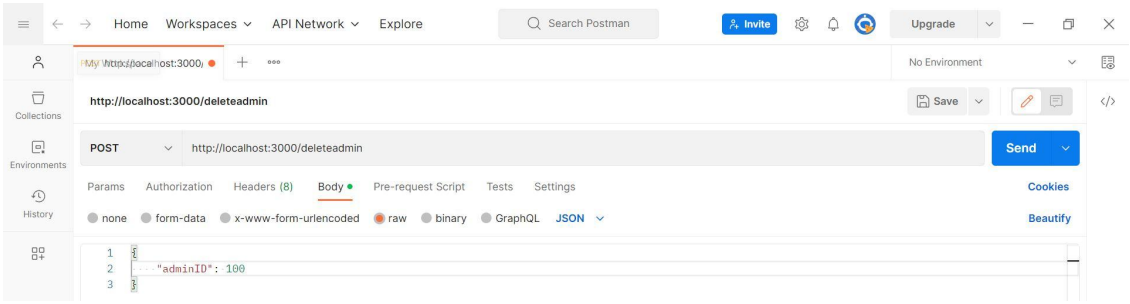
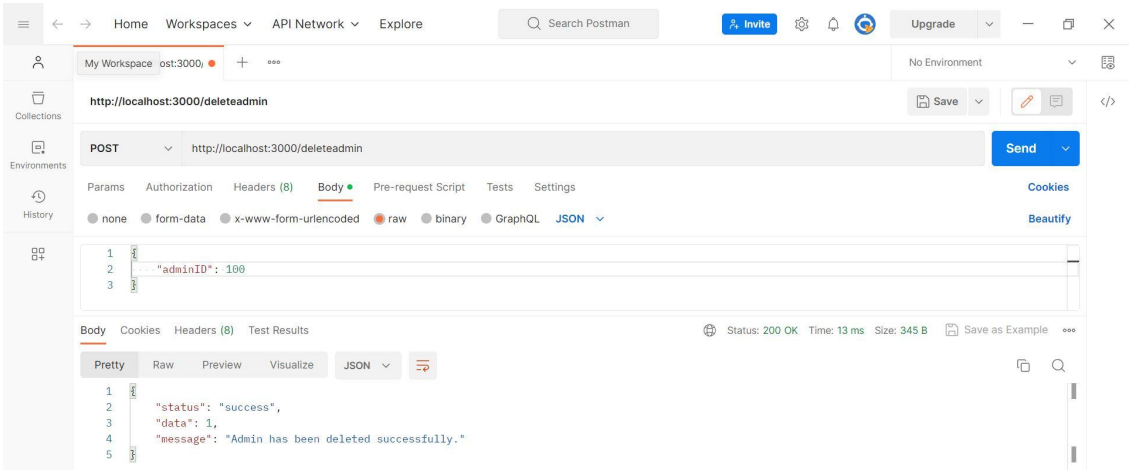
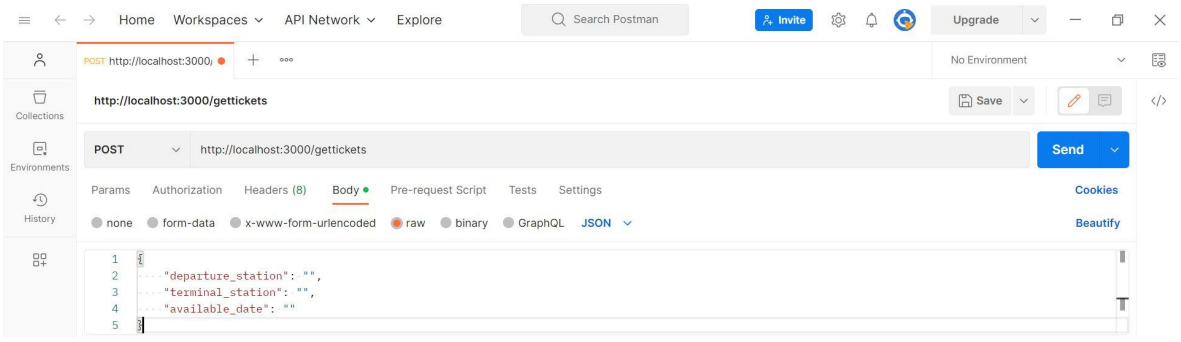
Administrator Delete

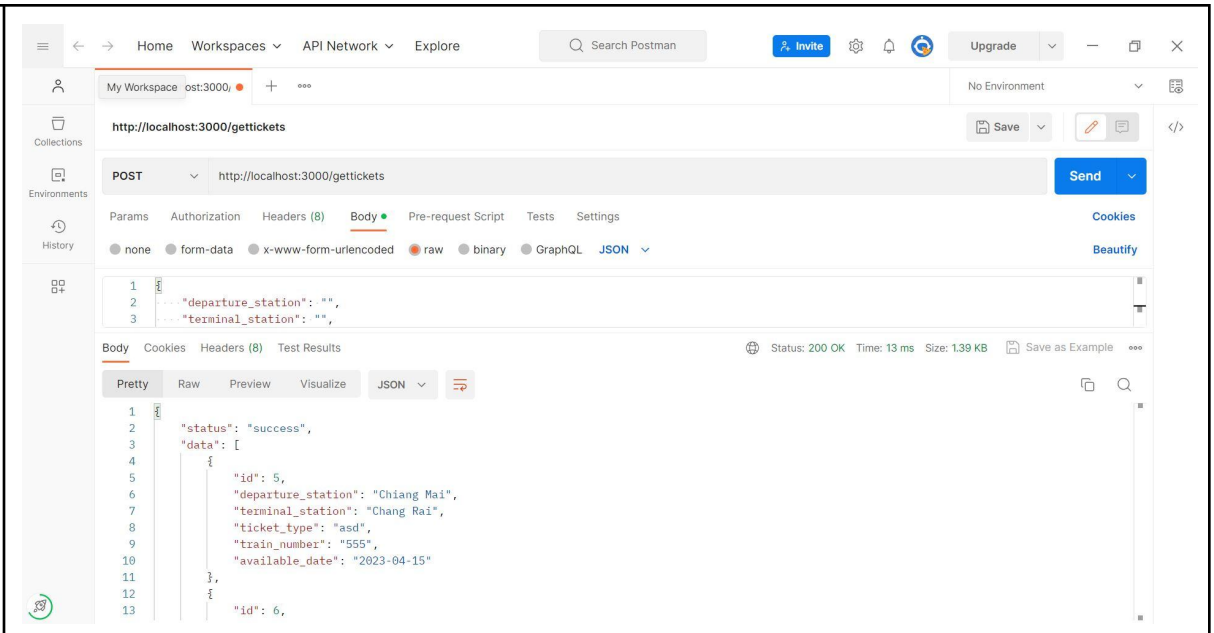
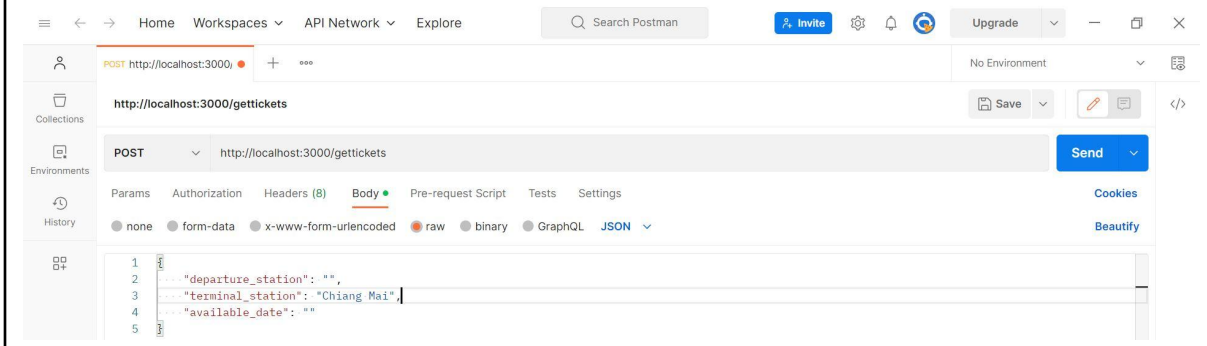
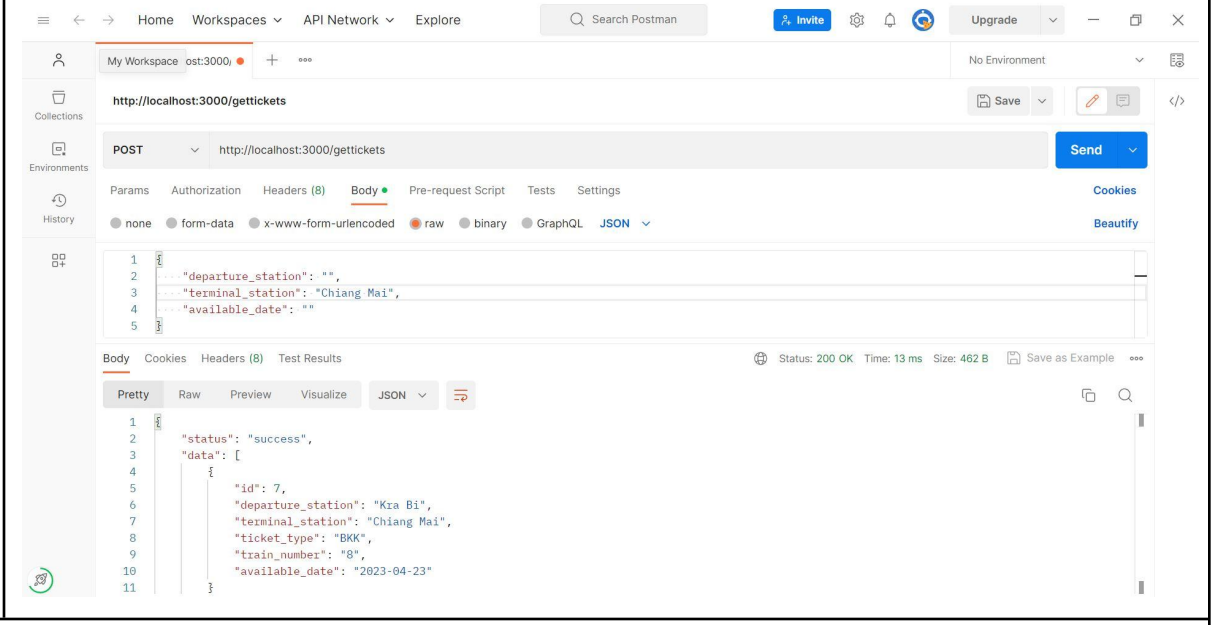
Deleteadmin

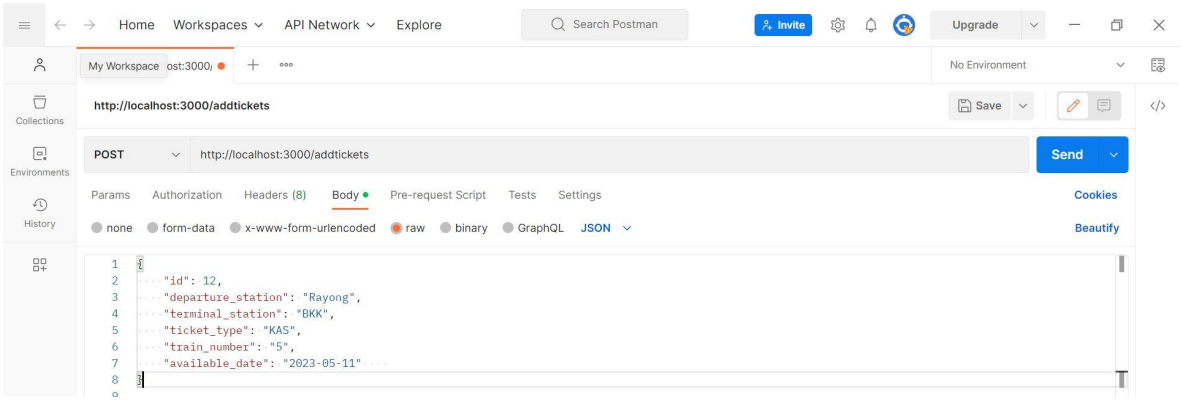
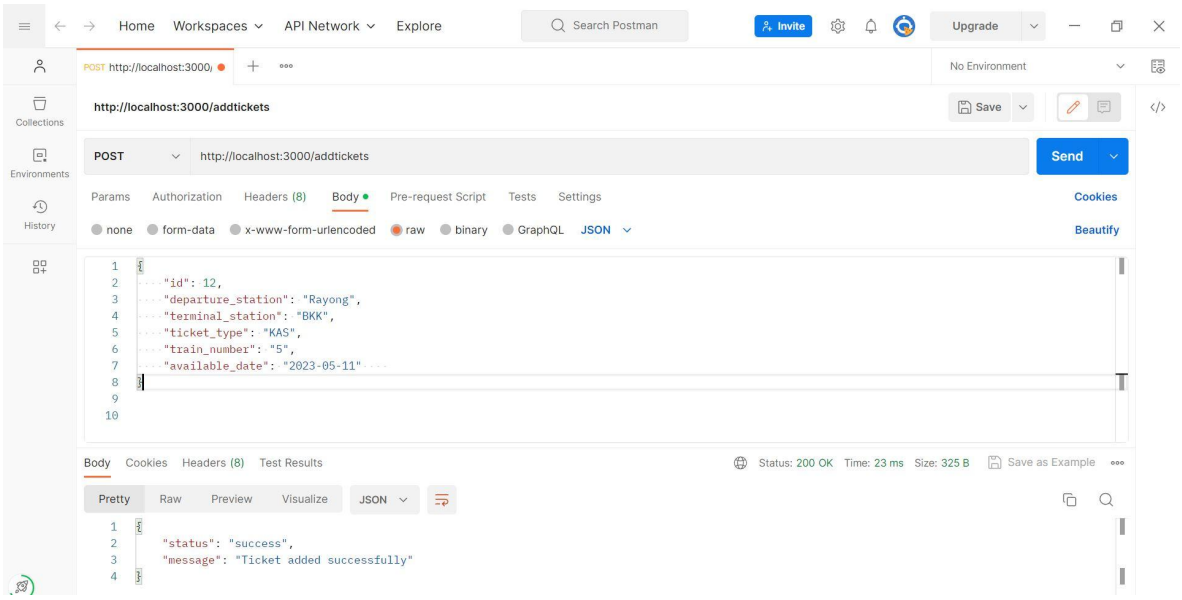
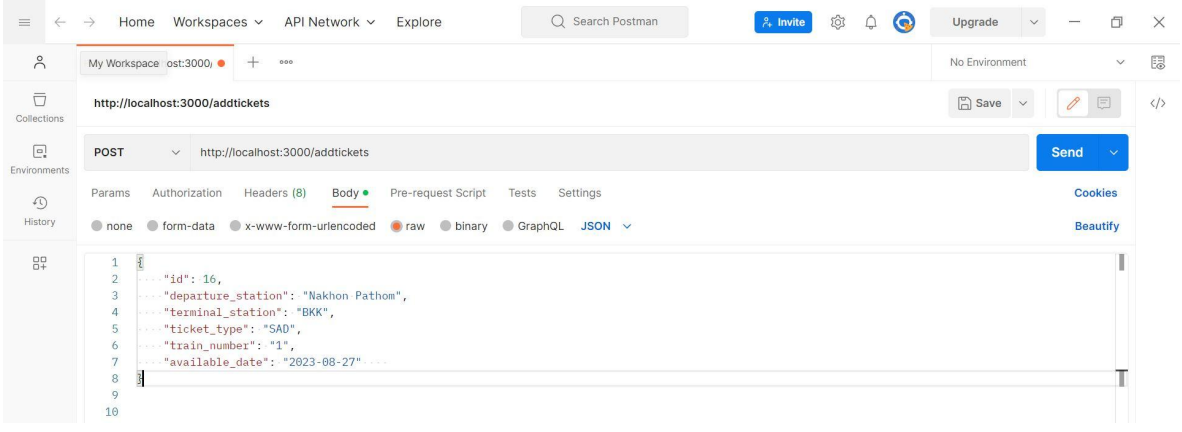


Test case1

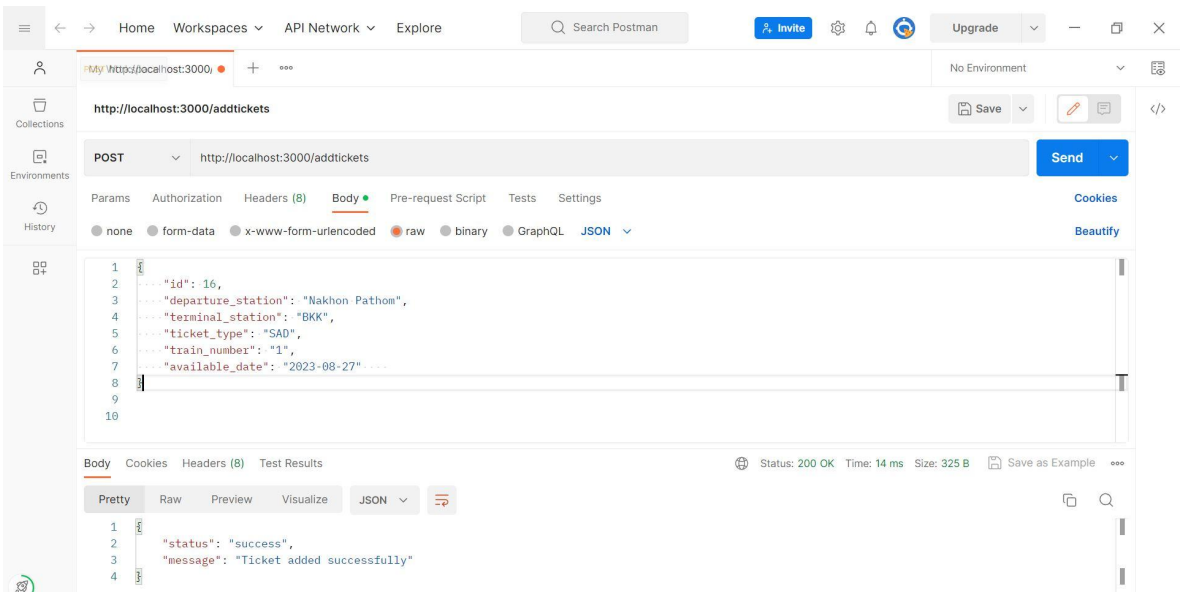


Deleteadmin	
Test case2	
Product Service for administrators	
Product Search/View	
Gettickets	

Test case1	
Gettickets	
Test case2	
Product Insert	

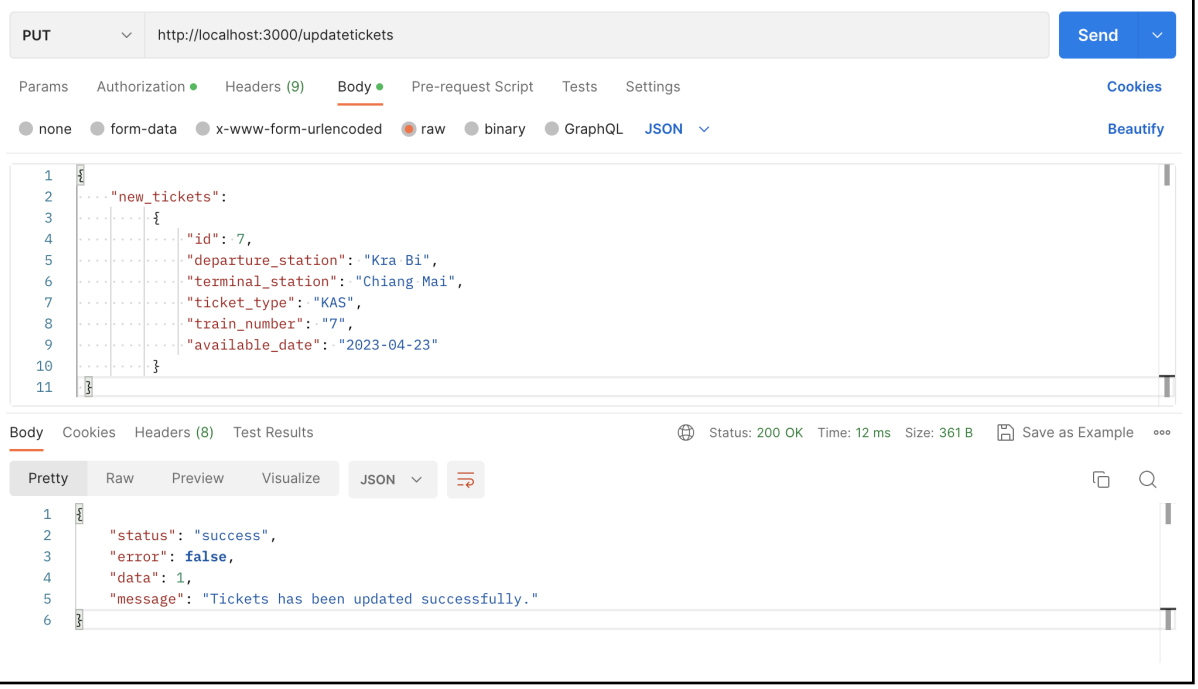
Addtickets	
Test case1	
Addtickets	

Test case2

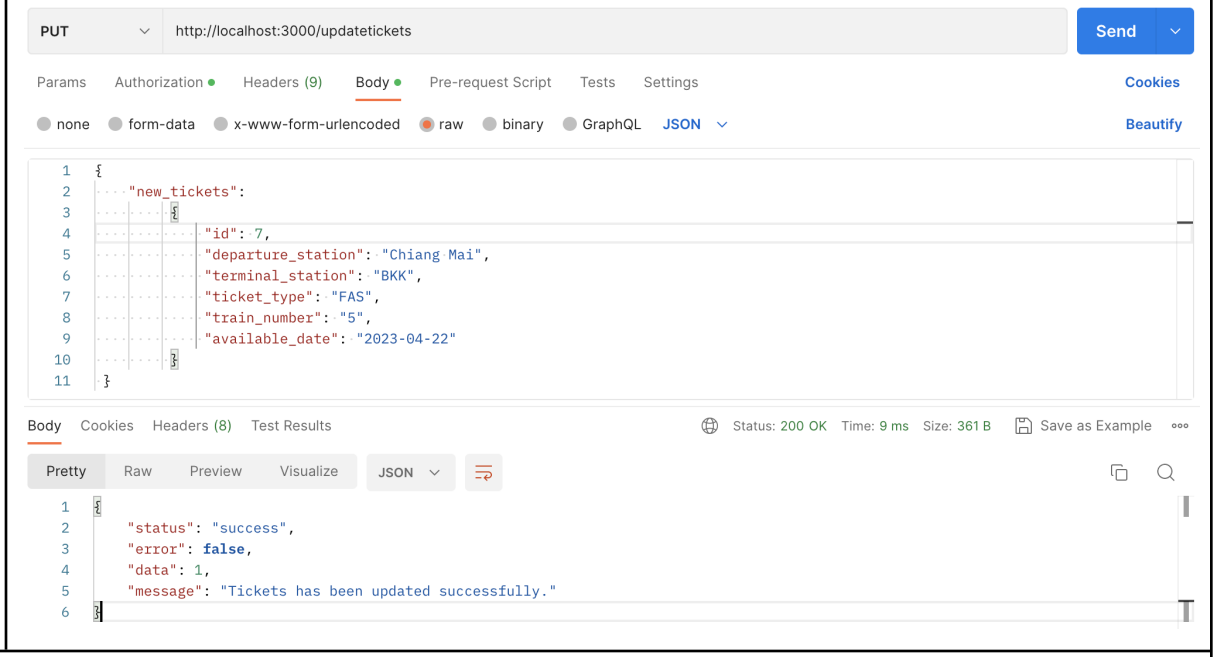


Product Update: update ticket information by ticket ID.

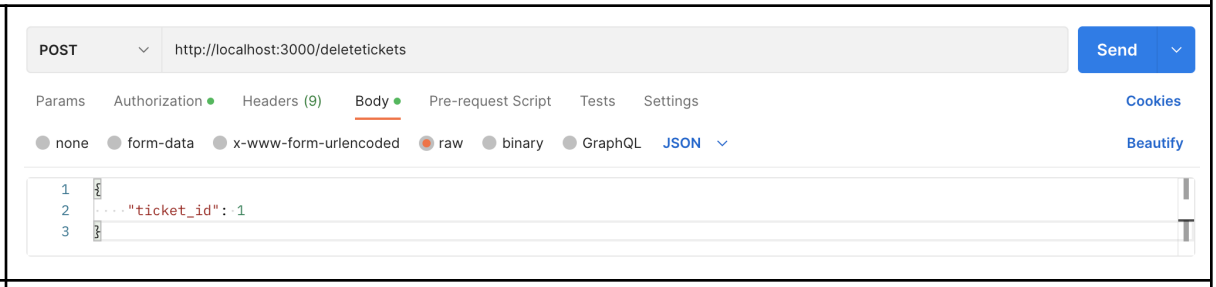
Test case1



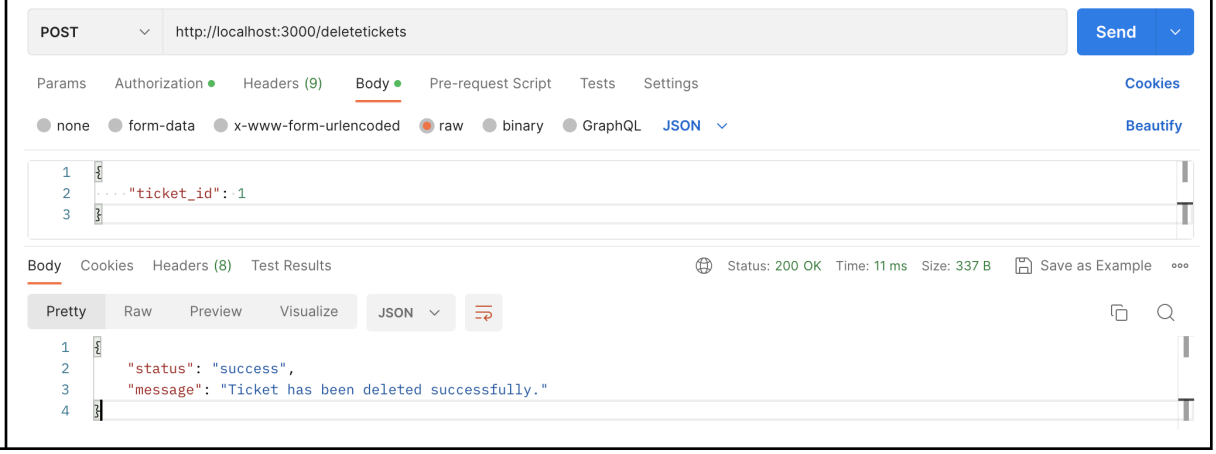
Test case2

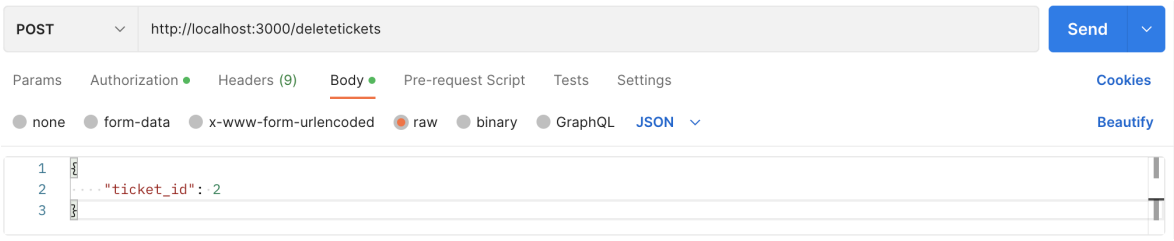
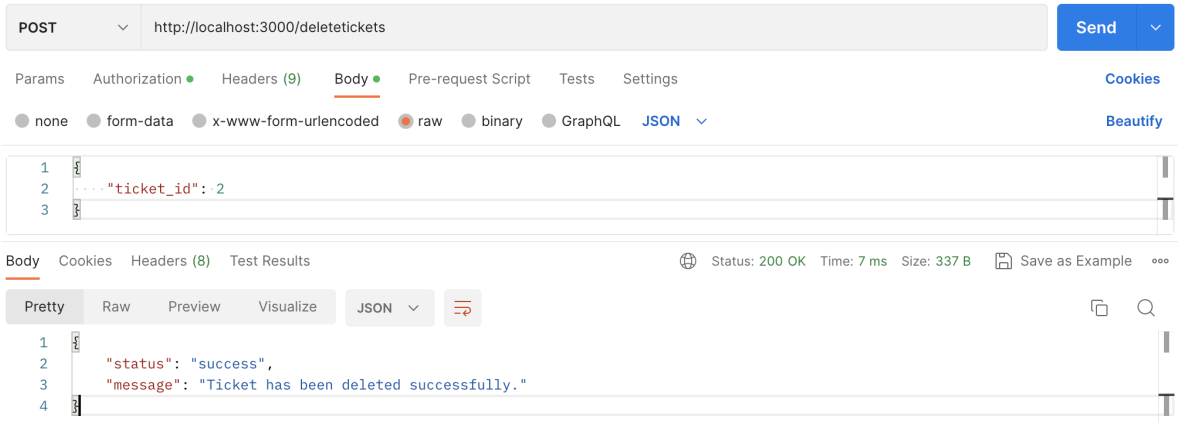


Product Delete: delete the ticket with specific ticket ID



Test case1



	 POST http://localhost:3000/deletetickets Send Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify 1 { 2 "ticket_id": 2 3 }
Test case2	 POST http://localhost:3000/deletetickets Send Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify 1 { 2 "ticket_id": 2 3 } Body Cookies Headers (8) Test Results Status: 200 OK Time: 7 ms Size: 337 B Save as Example Pretty Raw Preview Visualize JSON 1 { 2 "status": "success", 3 "message": "Ticket has been deleted successfully." 4 }

Appendix

Appendix A

```
function search(){
   let data = {
       "departure_station": "",
       "terminal_station": "",
       "available_date": ""
   };
   fetch("http://localhost:3000/gettickets", {
       method: "POST",
       headers: {
           "Content-Type": "application/json"
       },
       body: JSON.stringify(data)
   })
   .then(response => response.json())
   .then(res => {
       console.log(res.data);
       let destation = document.getElementById("destation").value;
       let terstation = document.getElementById("terstation").value;
       let datesearch = document.getElementById("datesearch").value;

       let search_result = document.getElementById("search_result");
       search_result.innerHTML = `
           <article class="headforsearchresult">
               <h1> Search Result </h1>
           </article>
       `
       let data_result = "";
       //console.log(res.data)
       let count = 0;
       for (let i = res.data.length-1; i >= 0; i--) {
           //console.log(destation)
           //console.log(res.data[i].departure_station)
```

```
            if (destation != "" && res.data[i].departure_station != destation){
                continue;
            }
            if (terstation != "" && res.data[i].terminal_station !=
terstation){
                continue;
            }
            if (datesearch != "" && res.data[i].available_date != datesearch){
                continue;
            }
            count++;
            data_result += `
                <article class="column">
                    <h2>Ticket ${count}</h2>
                    <a href="/detail1" class="readmoredetail"> Read more detail
</a>
                    <section class="dummy_text">
                    ${res.data[i].departure_station} >>
${res.data[i].terminal_station}
                    <br> Date: ${res.data[i].available_date}
                    <article class="dummy_box">
                    <form class="contain_row">
                        <div class="row inline">
                        Departure Station <br> ${res.data[i].departure_station}
                        </div>
                        <div class="row inline">
                        Terminal Station <br> ${res.data[i].terminal_station}
                        </div>
                        <div class="row inline">
                        Type <br> ${res.data[i].ticket_type}
                        </div>
                        <div class="row inline">
                        Train <br> ${res.data[i].train_number}
                        </div>
                        <div class="row inline">
                        <a href="#" id="editbox"> Click for edit </a>
                        </div>
```

```
                    </form>
                    </article>
                </article>      `
        }
        if (count == 0){
            data_result = `
                <article class="column">
                    <h2> No result </h2>
                </article>
            `
        }
        search_result.innerHTML += data_result;
    })
    .catch((error) => {
        console.error('Error:', error);
    });
}
```

Appendix B

```
function getadmin(){
   let data = {
       "adminID": "",
       "admin_username": "",
       "admin_fullname": ""
   };

   fetch("http://localhost:3000/getadmin", {
       method: "POST",
       headers: {
           "Content-Type": "application/json"
       },
       body: JSON.stringify(data)
   })
   .then(response => response.json())
   .then(res => {
       console.log(res.data);
       let searchuser_results = document.getElementById("searchuser_results");

       let admin_id = document.getElementById("admin_id").value;
       let admin_username = document.getElementById("admin_username").value;
       let admin_fullname = document.getElementById("admin_fullname").value;

       let data_result = "";
       let count = 0;

       for (let i = 0; i < res.data.length; i++) {

           if (admin_id != "" && res.data[i].adminID != admin_id){
               continue;
           }
           if (admin_username != "" && res.data[i].admin_username !=
admin_username){
               continue;
           }
```

```
            if (admin_fullname != "" && res.data[i].admin_fullname !=
admin_fullname){
                continue;
            }
            count++;
            data_result += `<div class="user">
            <div class="sameline">

            <img
src="https://drive.google.com/uc?id=1IAzZJosZs_bSZ9aMXmaZxg6jdXlH2GMP"
width="50" height="50">
            <h4 style="color: black;">   ${res.data[i].admin_email}</h4>
            <h4 style="color: //#region ;">   Admin_fullname:
${res.data[i].admin_fullname} </h4>
            </div>

            <div class="right">
            <button class="edit"><a
onclick="updateadmin(${res.data[i].adminID})" href="/signup" >Edit</a></i>
            </button>
            <button class="delete"
onclick="deleteadmin(${res.data[i].adminID})"><a href="#" >Delete</a></i>
            </button>
            </div>
        </div>`
        }
        if (count == 0){
            data_result = `
                <div class="user">
            <div class="sameline">
            <h4>   No result</h4>
            </div>
        </div>`
        }
        searchuser_results.innerHTML = data_result;

    })
```

```
    .catch((error) => {
        console.error('Error:', error);
    });
}
```

Appendix C

```
function addadmin(){
   let admin_fullname = document.getElementById("admin_fullname").value;
   let admin_username = document.getElementById("admin_username").value;
   let admin_email = document.getElementById("admin_email").value;
   let admin_phonenum = document.getElementById("admin_phonenum").value;
   let admin_role = document.getElementById("admin_role").value;
   let admin_address = document.getElementById("admin_address").value;
   let admin_password = document.getElementById("admin_password").value;
   let admin_confpassword =
document.getElementById("admin_confpassword").value;

   if (admin_password != admin_confpassword){
       alert("Password not match");
       return;
   }
   if (admin_password.length > 8){
       alert("Password max length is 8")
       return;
   }
   fetch("http://localhost:3000/add_admin", {
       method: "POST",
       headers: {
           "Content-Type": "application/json"
       },
       body: JSON.stringify({
           admin_fullname: admin_fullname,
           admin_username: admin_username,
           admin_email: admin_email,
           admin_phonenum: admin_phonenum,
           admin_role: admin_role,
           admin_address: admin_address,
           admin_password: admin_password,
           admin_confpassword: admin_password
       })
   })
```

```
    .then(response => response.json())
    .then(res => {
        console.log(res.data);
        if (res.status == "success"){
            //reload page
            alert('Add admin success');
            window.location.href = "/usermanage";
        }
        else {
            alert("Error")
        }
    }
    )
}
```