Конкурсное задание

Республиканского конкурса профессионального мастерства WorldSkills Kazakhstan 2024

по компетенции Веб технологии

Module B: REST API Places & Routes

Адаптировали:

Главный эксперт — Попов Денис

Международный эксперт — Зияданова Айжан

CONTENTS

INTRODUCTION

Time to complete: 3 hours

"Nomad Tour" is a local startup company. They would like to create a website that could help users to get itinerary to reach their destination place by schedule. They would like to create a website that could help users to get itinerary to reach their destination place by schedule. Nomad Tour is an on-demand hop on hop off in a bus around Astana.

You pay once upfront to hop on hop off all day (8:30AM-6PM) between designated meeting points by using our website to call a bus to each point at their convenience. The user can set their start and end place, then the system will find the fastest route(s) between the two given stations depending on the lines running. The self-guided website also contains helpful information of nearby attractions and local restaurants.

PROJECT DESCRIPTION AND TASKS

The description for the first phase of the project is listed below. The first task is to create a restful web service API that can be used by the front end to communicate the data.

Web Service

"Nomad Tour" will provide the list of web services that need to be created. Web service specification will contain the URL path of web service, request method, requested parameter on URL, requested parameter on body request, response result and response status. Request and response on web service should only contain JSON.

There are three roles/types of users: public, authenticated user and admin.

These are the list of web service that requested by the company:

1. **Authentication**
    a. **Login** (v1/auth/login)
    Description: For client to get login token via username and password
    Request method: **POST**
    Requested parameter:
    - Body:
        - username
        - password

    Response result:

    - If success,
        - header: response status: 200
        - body:
            - token: authorization token (to be valid until logout). Token will be generated by the system from logged in username with sha1 encryption method
            - role (ADMIN / USER)

- If username/password not correct or empty,
  - header: response status: 401
  - body: message: invalid login

b. Logout (v1/auth/logout)

Description: For server to invalid the user's token
Request method: **GET**
Header: authorization bearer token
Response result:

- If success,
  - header: response status: 200
  - body:
    - message: logout success
- If unauthorized user access it, data:
  - Message: Unauthorized user
  - Response status: 401

2. Place

a. All Places (v1/place)

Description: For client to list all places in the database (include user's search history indexed based on the frequency)
Request method: **GET**
Header: authorization bearer token
Response result:

- Body:
  - All data on array; consists of id, name, type, latitude, longitude, x, y, image_path, description, open_time, close_time.
- Response status: 200
- If unauthorized user access it, data:
  - Message: Unauthorized user
  - Response status: 401

b. Find Place (v1/place/{ID})

Description: For client to fetch one place object via place ID.
Request method: **GET**
Header: authorization bearer token
Response result:

- Body:
  - object; property consists of name, type, x, y, image_path, open_time, close_time, description, num_searches
- Response status: 200

c. **Create place** (v1/place), only admin can access this API
Description: For client to create a new place object. Image file from client must be uploaded to server. You can use form data to upload an image. You must use library Poi.php or Poi.js from media file to calculate x, y from latitude and longitude
Request method: **POST**
Header: authorization bearer token
Request parameter:

- Body:
  - name
  - type
  - latitude
  - longitude
  - image
  - open_time
  - close_time
  - [description]

Response result:

- If success, body:
  - Message: create success
  - Response status: 200
- If failed, body:
  - Message: Data cannot be processed
  - Response status: 422
- If unauthorized user access it, body:
  - Message: Unauthorized user
  - Response status: 401

d. **Update place** (v1/place/{ID}), only admin can access this API
Description: For client to update an existing place object via given place ID. If an image file is provided, it must be uploaded to server, old related image (if any) must be deleted. (Use library Poi.php or Poi.js to calculate x, y from latitude and longitude)
Request method: **PUT/PATCH**
Header: authorization bearer token
Request parameter:

- Body:
  - [name]
  - [type]
  - [latitude]
  - [longitude]
  - [image]
  - [open_time]
  - [close_time]
  - [description]

Response result:

- If success, body:
  - Message: update success
  - Response status: 200
- If failed, body:
  - Message: Data cannot be updated
  - Response status: 400
- If unauthorized user access it, body:
  - Message: Unauthorized user
  - Response status: 401

e. **Delete place** (v1/place/{ID}), only admin can access this API

Description: A request to delete a place object via given place ID. Related image and schedule (if any) must be deleted

Request method: **DELETE**

Header: authorization bearer token

Response result:

- If success, body:
  - Message: delete success
  - Response status: 200
- If failed, body:
  - Message: Data cannot be deleted
  - Response status: 400
- If unauthorized user access it, data:
  - Message: Unauthorized user
  - Response status: 401

3. **Schedule**

a. **All Schedules** (v1/schedule), only admin can access this API

Description: For client to list all schedules in the database

Request method: GET

Header: authorization bearer token

Response result:

- body:
  - All data on array; Each element consists of: line, from_place (name), to_place (name), departure_time, arrival_time, distance, speed, status.
  - Response status: 200
  - If unauthorized user access it, data:
    - Message: Unauthorized user
    - Response status: 401

b. **Create schedule** (v1/schedule), only admin can access this API

Description: For client to create a schedule in database. A schedule describes when and where attractions and local restaurants and departs from one stop and arrive at the next stop.

Request method: **POST**
Header: authorization bearer token
Request parameter:

- Body:
  Object: consisting of line, from_place_id, to_place_id, departure_time, arrival_time, distance, speed (arrival_time must be after departure_time; the times must be in range: 08:30AM – 06:00PM)

Response result:

- If success,
  - header: response status: 200
  - body: message: create success
- If failed,
  - header: response status: 422
  - body: message: Data cannot be processed
- If unauthorized user access it,
  - header: response status: 401
  - body: message: Unauthorized user

c. Update schedule (v1/ schedule /{ID}), only admin can access this API
Description: For client to update an existing schedule object via given schedule ID.
Request method: **PUT/PATCH**
Header: authorization bearer token
Request parameter:

- Body:
  - [line]
  - [from_place_id]
  - [to_place_id]
  - [departure_time]
  - [arrival_time]
  - [distance]
  - [speed]

Response result:

- If success, body:
  - Message: update success
  - Response status: 200
- If failed, body:
  - Message: Data cannot be updated
  - Response status: 400
- If unauthorized user access it, body:
  - Message: Unauthorized user
  - Response status: 401

d. **Delete schedule** (v1/schedule/{ID}), only admin can access this API
Description: A request to delete an existing schedule via given schedule ID.
Request method: **DELETE**
Header: authorization bearer token
Response result:
- If success,
  - header: response status: 200
  - body: message: delete success
- If unauthorized user access it,
  - header: response status: 401
  - body: message: Unauthorized user

4. **Route**

a. **Route Search**
(v1/route/search/{FROM_PLACE_ID}/{TO_PLACE_ID}/[DEPARTURE_TIME])
Description: A request to fetch multiple route suggestions to depart from a given stop (departure/source) and arrive at another stop (destination/target). By default, the search uses current server time. It also allows an optional departure time to override the default server time. The search should search the routes that depart from the given place at specific time and arrive the destination place, sorting by the earliest arrival time and limited to 5 routes result.
Only return available schedules.
Request method: **GET**
Header: authorization bearer token
Response result:
- If success, data:
  - Array of routes. Each route contains:
    - Number of history selection of this route.
    - Array of schedules:
      - id
      - line
      - departure_time
      - arrival_time
      - travel_time
      - from_place; consist of id, name, type, longitude, latitude, x, y, open_time, close_time, description, image_path
      - to_place; consist of id, name, type, longitude, latitude, x, y, open_time, close_time, description, image_path
  - Response status: 200
- If unauthorized user access it, body:
  - Message: Unauthorized user
  - Response status: 401

b. **Store Route Selection History** (v1/route/selection)

Description: For client to save a user selected route into the system.

Request method: **POST**

Header: authorization bearer token

Request parameter:

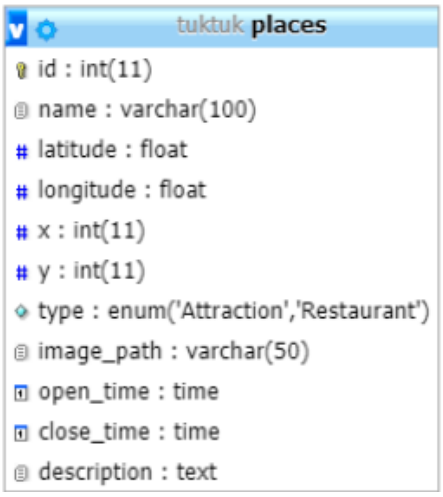
- Body:
  - from_place_id
  - to_place_id
  - schedule_ids, array of schedule_id for the route

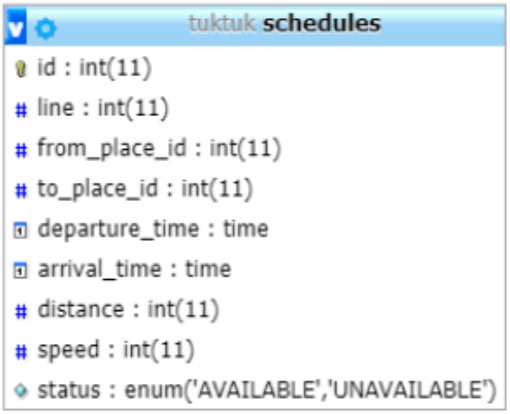
Response result:

- If success, body:
  - Message: create success
  - Response status: 200
- If failed, body:
  - Message: Data cannot be processed
  - Response status: 422
- If unauthorized user access it, body:
  - Message: Unauthorized user
  - Response status: 401

DATABASE

The database (to be normalized to third normal form and using reference constraints where appropriate), create at least the default tables, specified as follows (data provided in CSV):





Notes

- [key/item] with bracket [] is optional. Item with braces {} is mandatory.
- The API needs to be implemented as specified, but you can add optional fields.
- Competitors should implement a server-side framework/library that are provided.
- The specified database tables need to be implemented. More tables may be added if needed. Provide a final SQL-dump and ERD screen as specified below. All API should fulfill all requirements as stated in the description. All prefix, RESTful-URL and HTTP Method from given API link should be implemented correctly and not be changed. If needed, you may add other API, besides all API that already mentioned in this document.
- Create the following users to login to the system:
    - Admin with username: admin and password: adminpass,
    - User1 with username: user1 and password: user1pass,
    - User2 with username: user2 and password: user2pass

INSTRUCTIONS FOR THE COMPETITOR:

Follow these instructions to submit your work:

Save work on the desktop or in the folder with domains in OpenServer:

1. Save and work on your API in a directory ‘moduleB-XX’, where ‘XX’ should be number your workplace.
2. ERD screen shot named “XX_ERD.png” in “db-dump” folder inside of root project
3. Database dump named “XX_database.sql” in “db-dump” folder inside of of root project

MARKING SCHEME

	Description	**Assessment**
B1	General	1.75
B2	Database	3.25
B3	Auth	2.00
B4	Places	6.00
B5	Schedule	5.00
B6	Route	4.00
	Total	22.00