# SERVER SIDE

**INTRODUCTION**

“**Travel Blitar**” is a local startup company. They would like to create a website that could help users to get itinerary of public transportation to reach their destination place by schedule. They will provide the coordinate for train and bus station. The user can set their start and end place/station, then the system will find the fastest route(s) between the two given stations depending on the vehicles/lines running. The routes may include transfers between multiple vehicles/lines. You can use your own algorithm to solve the problem. The system should help the user to decide which route is faster based on the public transportation schedule.

The system should be separated by client and server architecture. The customer is asking for a web service architecture. Development phase should be separated into two phases. The **first phase** is creating the **backend web service**. You can use and enhance the design that is given with a client side framework to communicate with the services.

Glossary:

Schedule: is a moving from one station/place to the next station/place at specific times.

Line: consists of the multiple schedules and run by a vehicle (for example the any color of the

line).

Route: is the trip for a passenger from departure/source to destination/target.

## Description of project 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

“**Travel Blitar**” 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:

#### Authentication

* + 1. Login (*v1/auth/login*)

Description: For client to get login token via username and password Request method: **POST**

Header: header authorization basic 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 md5 encryption method

Role (ADMIN / USER)

* + - * If username/password not correct or empty,
        + header: response status: 401
        + body: message: invalid login
    1. Logout (*v1/auth/logout?token={AUTHORIZATION\_TOKEN}*) Description: For server to invalid the user’s token

Request method: **GET**

Header: header authorization basic Response result:

* + - * If success,
        + header: response status: 200
        + body:

message: logout success

* + - * If unauthorized user access it, data:
        + Message: Unauthorized user
        + Response status: 401

#### Place

* + 1. All Places (*v1/place?token={AUTHORIZATION\_TOKEN}*)

Description: For client to list all places in the database

Request method: **GET**

Header: header authorization basic

Response result:

body:

* All data on array; consists of id, name, latitude, longitude, image\_path, description.
* Response status: 200
* If unauthorized user access it, data:
  + Message: Unauthorized user
  + Response status: 401
    1. Find Place (*v1/place/{ID}?token={AUTHORIZATION\_TOKEN}*)) Description: For client to fetch one place object via place ID. Request method: **GET**

Header: header authorization basic Response result:

* + - * body:
        + object; property consists of id, name, latitude, longitude, image\_path, description.
        + Response status: 200
    1. Create place (*v1/place?token={AUTHORIZATION\_TOKEN}*), only admin can access this API

Description: For client to create a new place object. Image file from client should be uploaded to server. You can use form data to upload an image.

Request method: **POST**

Header: header authorization basic Request parameter:

* Body:
  + name
  + latitude
  + longitude
  + image
  + [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
    1. Delete place (*v1/place/{ID}?token={AUTHORIZATION\_TOKEN}*), only admin can access this API

Description: A request to delete a place object via given place ID. Request method: **DELETE**

Header: header authorization basic 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
    1. Update place (*v1/place/{ID}?token={AUTHORIZATION\_TOKEN}*), 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 should be uploaded to server.

Request method: **POST**

Header: header authorization basic Request parameter:

* Body:
  + [name]
  + [latitude]
  + [longitude]
  + [image]
  + [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

#### Schedule

* + 1. Create schedule (*v1/schedule?token={AUTHORIZATION\_TOKEN}*), only admin can access this API

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

Request method: **POST**

Header: header authorization basic Request parameter:

* Body:
  + Object: consisting of type (bus or train), line, from\_place\_id, to\_place\_id, departure\_time, arrival\_time, distance (km), speed (hour)

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
    1. All Schedule (*v1/schedule?token={AUTHORIZATION\_TOKEN}*)

Description: For client to show schedule in database. A schedule describes when and where a bus/train departs from one stop and arrive at the next stop.

Request method: **GET**

Header: header authorization basic

Response result:

body:

* All data on array; consisting of type (bus or train), line , from\_place\_id, to\_place\_id, departure\_time, arrival\_time, distance, speed.
* Response status: 200
* If unauthorized user access it, data:
  + Message: Unauthorized user
  + Response status: 401
    1. Delete schedule (*v1/schedule/{ID}?token={AUTHORIZATION\_TOKEN}*), only admin can access this API

Description: A request to delete an existing schedule via given schedule ID. Request method: **DELETE**

Header: header authorization basic 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

#### Route

* + 1. Route Search (*v1/route/search/{FROM\_PLACE\_ID}/{TO\_PLACE\_ID}?token={A UTHORIZATION\_TOKEN}*)

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.

The route allows transfer to different bus/train at the same station. All transfer happens at the same stop. There is no walk and no minimum transfer time required.

Request method: **GET**

Header: header authorization basic Response result:

* If success, data:
  + Array of routes. Each route contains :
    - Number of history selection of this route.
    - Array of schedules:
      * id
      * type
      * line
      * departure\_time
      * arrival\_time
      * travel\_time
      * from\_place; consist of id, name, longitude, latitude, x, y, description, image\_path
      * to\_place; consist of id, name, longitude, latitude, x, y, description, image\_path

o Response status: 200

* + - * If failed, data:
        + Message: Unauthorized user
        + Response status: 401
    1. Store Route Selection History (*v1/route/selection?token=[AUTHORIZATION\_TOKEN]*) Description: For client to save a user selected route into the system.

Request method: **POST**

Header: header authorization basic Request parameter:

* + - * Body:
        + from\_place\_id
        + to\_place\_id
        + schedule\_id, 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

Website components that have been provided are:

#### Search Route

Functionalities:

* + 1. Selecting *from* place and *to* place
* Fetch the list of places from the service. Sorted alphabetically by default. Places used by this user (if logged in) will be at top of list (based on frequency).
  + 1. Searching routes
* Search the routes by using *from (source)* and *to (target)* input.

#### Routes List

After a user search the routes, then this component will show:

Search results on the left, schedules for this route will appear

As same as the API specification describes: The search should search the routes that depart from the given place at specific time and arrive to the destination place, sorting by the earliest arrival time and limited to 5 routes result.

The route allows transfer to different bus/train at the same station. A transfer happens at the same stop. There is no walk and no minimum transfer time required.

The list of train or bus schedule from the result, consist of:

* Numbering.
* Time Schedule **(Departure time at from place, Arrival time at to place)**.
* Total travel time
* Number of transfers/changes of line
* Number of selection on this routes by all users (more on that later in 4. Search history).
* This is the example scenario for search results, assuming from A to B that departs at 13:00:
  + A to B => departure time -> 13:00:00 arrival time -> 13:14:00, Bus Line 3, 14 minutes, 0 transfers.
  + A to B => departure time -> 13:02:00 arrival time -> 13:22:00, Bus Line 2, Train Line 4, 22 minutes, 1 transfer
  + A to B => departure time -> 13:01:00 arrival time -> 13:35:00, Bus Line 2, Bus Line 1, Bus Line 5, 36 minutes, 2 transfer

### Route(s) selection history

The system stores all the routes that all user selected.

* + 1. The system stores all user’s selection.

#### User Authentication

Functionalities:

* + 1. Login and logout should happen on the same page without redirect.
    2. Login
* Show the login modal, after user click login link.
* On the login dialog there will be inputs for username and password.
* After the user logged in, the login link will be changed to logout link and the current username will be displayed besides the logout link.
* There will be role for the two types of authenticated user: if user is an admin the admin menu will be shown.
* The username entered and token received, token will be kept on the client for further requests, also after page refresh.
  + 1. Logout
* The display is reset: login link is shown, username and corresponding functionality disappear.

#### Notes

* Competitors should implement the server-side that are provided.
* Show error/feedback messages based on response from API.
* 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