

TEST PROJECT WEB DESIGN AND DEVELOPMENT

Server Side

WSC2017_TP17_ServerSide_actual

Submitted by:

Hantze Sudarma ID [Lead Expert]

Ilya Belyakov RU

Diyaa BenKhadra AE

Seung Lyul Ryu KR

Settachok Saennam TH

Manuel Schaffner CH

Chorng-Shiuh Koong TW

Zoltán Sisák HU

Bin Mu CN

Zhanar Kubzhasarova KZ

Jaafar Mohammed Almoadhen BH

Competition Time:

Part I - 3 hours

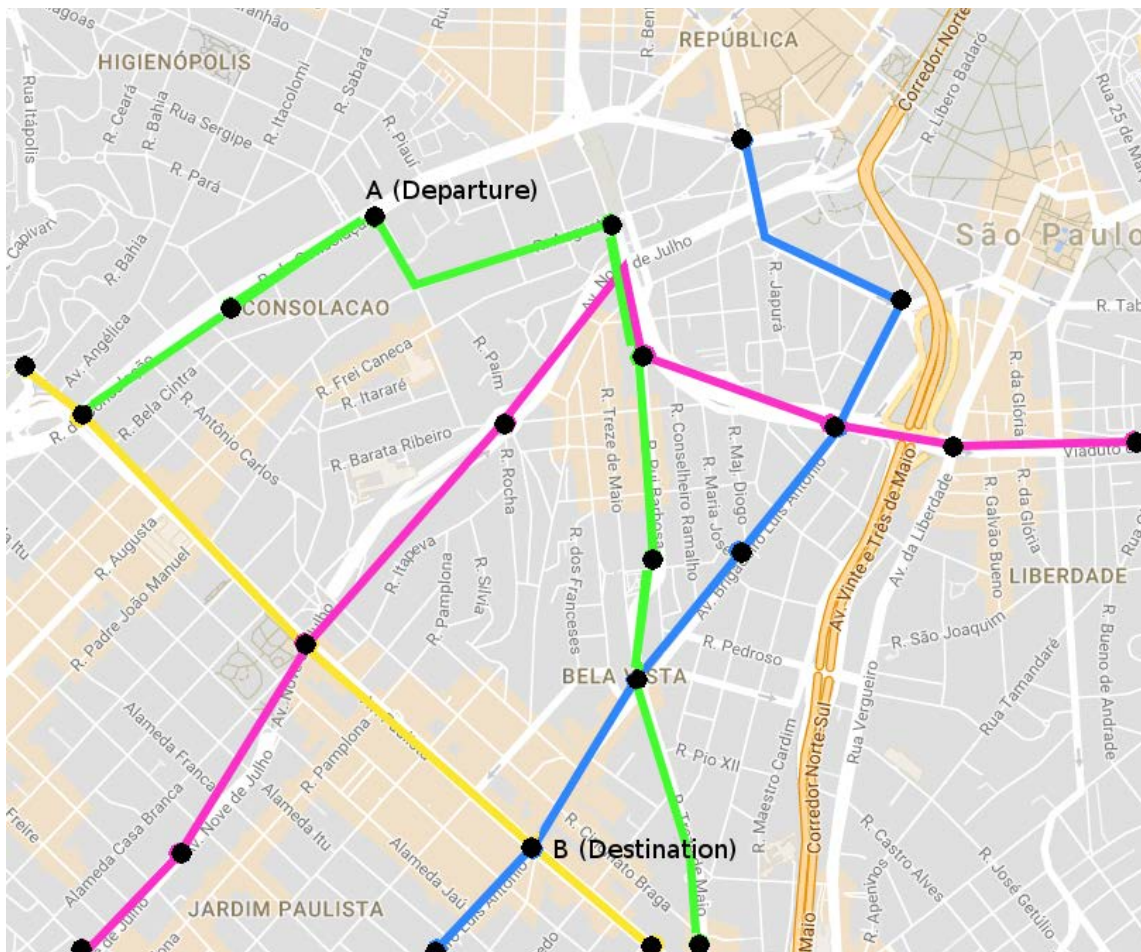
Part II - 3 hours





INTRODUCTION

“Bani Yas” 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 and a map of Abu Dhabi. 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** and the **second phase** is creating the **front end**. “Bani Yas” has made an initial design of the website to be used for the front-end development phase. 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.

I. Web Service

“Bani Yas” 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**

Header: header authorization basic

Requested parameter:

- Body:

- o Username
- o password

Response result:

- If success,
 - o header: response status: 200
 - o 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,
 - o header: response status: 401
 - o body: message: invalid login

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

2. Place

a. All Places (*v1/place?token={AUTHORIZATION_TOKEN}*)

Description: For client to list all places in the database (include user's search history indexed based on the frequency)

Request method: **GET**

Header: header authorization basic



Response result:

body:

- o All data on array; consists of id, name, latitude, longitude, x, y, image_path, description.
- o Response status: 200

- If unauthorized user access it, data:

- o Message: Unauthorized user
- o Response status: 401

b. 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:

- o object; property consists of id, name, latitude, longitude, x, y, image_path, description.
- o Response status: 200

c. 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:

- o name
- o latitude
- o longitude
- o x
- o y
- o image
- o [description]

Response result:

- If success, body:

- o Message: create success
- o Response status: 200

- If failed, body:

- o Message: Data cannot be processed
- o Response status: 422

- If unauthorized user access it, body:

- o Message: Unauthorized user
- o Response status: 401



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

- e. 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:
 - o [name]
 - o [latitude]
 - o [longitude]
 - o [x]
 - o [y]
 - o [image]
 - o [description]

Response result:

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



3. Schedule

- a. 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:
 - o Object: consisting of type (bus or train), line , from_place_id, to_place_id, departure_time, arrival_time, distance, speed

Response result:

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



4. Route

a. Route Search

(v1/route/search/{FROM_PLACE_ID}/{TO_PLACE_ID}/[DEPARTURE_TIME]?token={AUTHORIZATION_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:
 - o 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:
 - o Message: Unauthorized user
 - o Response status: 401

b. 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:
 - o from_place_id
 - o to_place_id
 - o schedule_id, array of schedule_id for the route

Response result:

- If success, body:
 - o Message: create success
 - o Response status: 200
- If failed, body:
 - o Message: Data cannot be processed
 - o Response status: 422



Notes:

- [key/item] with bracket [] is optional. Item with braces {} is mandatory.
- The path will be prepended with your directory “**XX_Server_A**” which will give *http://<servername>/XX_Server_A/api/v1/<services>*, where XX is your country code.
- HTTP method tunneling via *_method* is allowed.
- The API needs to be implemented as specified, but you can add optional fields.

TESTING

You should use the JSON file “Sample_ServerSide_A.postman_collection.json” provided to test the API using POSTMAN Chrome Extension. Please see the description in each test for usage (eg. Change <server>, <XX>, basic auth, etc.).

The system should handle input of data and responses should follow the above specification.

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):

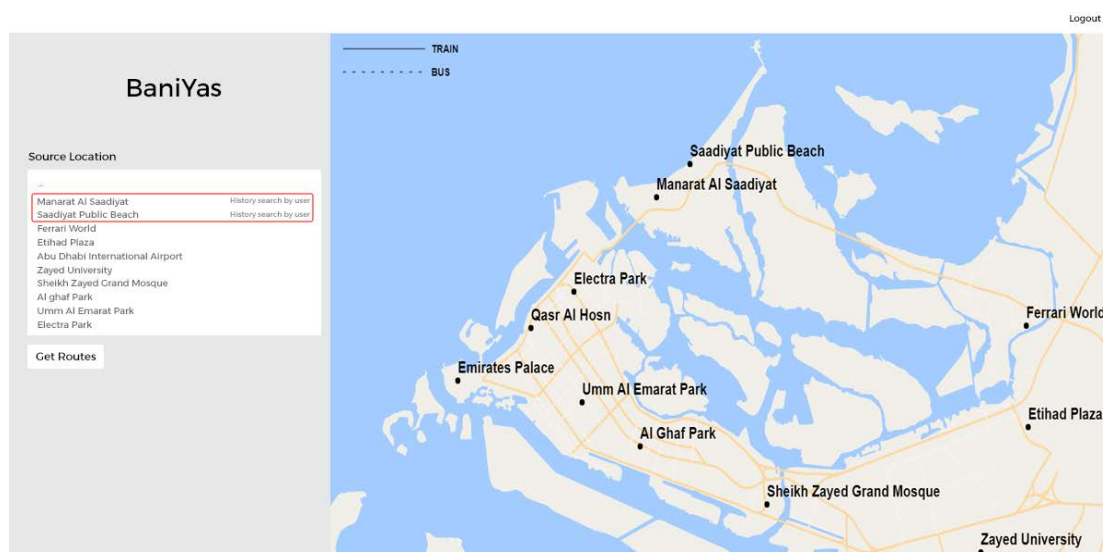
worldskills_2017_serverside_default_schedules	
id	int(11)
type	enum('TRAIN','BUS')
line	int(11)
from_place_id	int(11)
to_place_id	int(11)
departure_time	time
arrival_time	time
distance	int(11)
speed	int(11)
status	enum('AVAILABLE','UNAVAILABLE')

worldskills_2017_serverside_default_places	
id	int(11)
name	varchar(100)
latitude	float
longitude	float
x	int(11)
y	int(11)
image_path	varchar(50)
description	text

II. Front End

On the front-end there will be some functionality that is required by “**Bani Yas**”.

Create a front-end website for this company by using the provided single page template that is already prepared by the designer. Create all the functionalities for the page with communication with the backend web services API.





Website components that have been provided are:

1. Search Route

Functionalities:

- a. 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).
 - Selection with using auto complete (from database, not from browser history) for the *from* (*source*) and *to* (*target*) place. Auto complete will be filled based on user's history, every time user use this application then the system will save the data (*from/source* and *to/target*).
- b. Input departure time
 - Contain hour and minute. This is an optional field, doesn't required to be filled.
- c. Searching routes
 - Search the routes by using the *departure time (optional)*, *from (source)* and *to (target)* input.

2. Routes List

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

Search results on the left, schedules for this route will appear. By default, the search uses the current server time if user didn't fill in *departure time* before. If the user fills in departure time, then the search uses the given *departure time*.

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

3. Map View

The right side of the layout will show the map. Place coordinate will contain:

- a. There will be dot and place's name on map for each place based on the database records.
- b. If user clicks one of the dot, a floating box will be appeared near the dot. On floating box there will be a picture of the place, name of place and the short description about it. The box will be dismissed if user clicks the other area.
- c. Show a clearly visible dot (different from the normal place) on map for departure (from) place and destination (to) place.
- d. If user clicks the route on the result list, the route is shown on the map. Solid line for train, dashed line for bus.
- e. Each bus/train line should have a different color when drawing on the map.
- f. Show the legend for each vehicle line: different color for each line, and solid line for train, dashed line for bus.



4. Route(s) selection history

The system stores all the routes that all user selected.

- a. Whenever a user clicks on the route in the result list, this route is stored in the database.
- b. The system stores all user's selection.
- c. When a route shows in the result list, the total number of selection on the same route shows. Same route means the same from place and to place, regardless of departure time.

5. User Authentication

Functionalities:

- a. Login and logout should happen on the same page without redirect.
- b. 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.
- c. Logout
 - The display is reset: login link is shown, username and corresponding functionality disappear.

6. Admin Menu

Admin menu should only be shown after user login that has an admin role. Admin functionalities:

- a. Place
 - Create place
 - Update place
 - Delete place

Notes

- Competitors should implement a minimum of one of the server-side and client-side frameworks/libraries that are provided.
- The provided template design should be used, but it can be enhanced to get better functionality for your site.
- 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`
- Changes made in the data on the back-end server need to be propagated to the frontend. The data should be dynamically shown.
- Monitor screen size should not **affect** any function on the client side. (working on 1440px, 768px and 320px width)



INSTRUCTIONS TO THE COMPETITOR

Save the files of your application in directory on the server called "**XX_Server_A**" for the first phase and "**XX_Server_B**" for the second phase, where XX is your country code.

Files to be **collected** for the first phase on the server:

- Web service (**XX_Server_A**)
- ERD screen shot named "**XX_ERD.png**" in "db-dump" folder inside of **XX_Server_A**
- Database dump named "**XX_database.sql**" in "db-dump" folder inside of **XX_Server_A**

Files to be collected for the second phase on the server:

- Front-end website (**XX_Server_B**)
- Front-end website project files as inside folder **XX_Server_B**. You should ZIP all front-end development project files and name it **XX_Server_B.zip**.
- "**access.txt**" on the front-end root(http://competitorYY.wsad.local/XX_Server_B) to give information about the link/URL that should be **accessed** to mark the front-end. YY is the workstation number.

Web service will be marked in Postman Chrome Extension.

Front-end will be marked in Google Chrome.

FILES PROVIDED

ITEM	DESCRIPTION
Laravel PHP Framework	Web Design Competitor can choose which one of these frameworks he wants to use. Installation will be done on the familiarization-day (C-2)
Yii PHP Framework	Web Design Competitor can choose which one of these frameworks he wants to use. Installation will be done on the familiarization-day (C-2)
Angular2 / AngularJS	Web Design Competitor can choose which one of these frameworks he wants to use. Installation will be done on the familiarization-day (C-2)
Vue JS	Web Design Competitor can choose which one of these frameworks he wants to use. Installation will be done on the familiarization-day (C-2)
Media files	Web Contain: <ul style="list-style-type: none">- csv: schedule data, place location, image and description data. All in *.csv files.- database: table structure.- place_images: all images needed.- template-ui: default template.- mapinfo.txt: scalling map points.- Poi.php: reference for calculate map points.- Sample_Server_A.postman_collection.json.



MARKING SCHEME SUMMARY

SECTION	CRITERION	JUDGEMENT MARKS	MEASUREMENT MARKS	TOTAL
C1	Server Side API General	1.50	0.50	2.00
C2	Server Side DB setup	0	1.00	1.00
C3	Server Side Database and Auth	0.75	2.00	2.75
C4	Server Side API Places	0	2.00	2.00
C5	Server Side API Places Operation	0	2.25	2.25
C6	API Schedule	0	1.50	1.50
C7	Server Side API Route	0	2.00	2.00
D1	Server Side Client General	1.50	1.00	2.50
D2	Server Side Website design	2.50	0	2.50
D3	Server Side Interaction and Auth	1.25	3.25	4.50
D4	Server Side Search route	0	3.50	3.50
D5	Map view	0	3.00	3.00
D6	Admin menu	0	1.50	1.50
D7	Admin Places	0	1.00	1.00
Total		7.50	24.50	32.00