German International University
Dr. Amr Desouky
Eng. Donia Ali
Sarah Ashraf

last update 26/5/2023

Software Engineering
Project Specification

Due Date:
Backend 28th May
Frontend 6th June

1. View Endpoints

Your system is expected to have the following **User/Admin** shared HTML views:

- / ← Login
- /register
- /dashboard
- /resetPassword

Your system is expected to have the following **User** only HTML views:

- /subscriptions
- /tickets
- /prices
- /rides
- /requests/refund
- /requests/senior

Your system is expected to have the following **Admin** only HTML views:

- /manage/stations
- /manage/routes
- /manage/requests/refunds
- /manage/requests/seniors
- /manage/zones

2. Api Endpoints (Backend)

Your system is expected to support the following API endpoints which are broken down by views:

User API Endpoints by View

View	HTTP Method	HTTP Endpoint Route	Description	Request Body

/login	POST	/api/v1/users/login	Login user	{
				email: string,
				password: string
				}

/register	POST	/api/v1/users	Create User	{
				firstName: string,
				lastName:string,
				email:string,
				password: string,
				}

/dashboard	-	-	-	-

/resetPassword	PUT	/api/v1/password/reset	Reset Password	{
				newPassword: string
				}

/subscriptions	GET	/api/v1/zones	Get Zones Data	-

	POST	/api/v1/payment/subscription	Pay for subscription online	{
				,
				creditCardNumber:Integer,
				holderName:string,
				payedAmount:integer,
				subType:string,
				zoneId:integer
				}

/tickets	POST	/api/v1/payment/ticket	Pay for ticket online	{
				,
				creditCardNumber:Integer,
				holderName:string,
				payedAmount:integer,
				origin:string,
				destination:string.
				tripDate: dateTime
				}

	POST	/api/v1/tickets/purchase/subscription	Pay for ticket by subscription	{
				subId:integer,
				origin:string,
				destination:string.
				tripDate: dateTime
				}

/prices	GET	/api/v1/tickets/price/:originId & :destinationId	Check Price	-
/rides	-	-	-	-
/requests/refund	POST	/api/v1/refund/:ticketId	Refund Ticket	-

/requests/senior	POST	/api/v1/senior/request	Request Senior	{
				nationalId: integer
				}

/rises/simulate	PUT	/api/v1/ride/simulate	Simulate Ride	{
				origin:string,
				destination:string,
				tripDate:dateTime
				}

Admin API Endpoints by View

View	**HTTP Method**	**HTTP Endpoint Route**	**Description**	**Request Body**

/login	POST	/api/v1/users/login	Login user	{
				email: string,
				password: string
				}

/register	POST	/api/v1/users	Create User	{
				firstName:string,
				lastName: string,
				email:string,
				password:string,
				}

/dashboard	-	-	-	-

/resetPassword	PUT	/api/v1/password/reset	Reset Password	{
				newPassword:integer
				}

/manage/stations	POST	/api/v1/station	Create Station	{
				stationName: string
				}

/manage/stations	PUT	/api/v1/station/:stationId	Update Station	{
				stationName: string
				}

/manage/stations	DELETE	/api/v1/station/:stationId	Delete Station	-

/manage/routes	POST	/api/v1/route	Create Route	{
				newStationId: Integer,
				connectedStationId:integer,
				routeName:String
				}

/manage/routes	PUT	/api/v1/route/:routeId	Update Route	{
				routeName: string
				}

/manage/routes	DELETE	/api/v1/route/:routeId	Delete Route	-

/manage/requests/refunds	PUT	/api/v1/requests/refunds/:requestId	Accept/Reject Refund	{
				refundStaus:string
				}

/manage/requests/seniors	PUT	/api/v1/requests/senior/:requestId	Accept/Reject Senior	{
				seniorStaus:string
				}

/manage/zones	PUT	/api/v1/zones/:zoneId	Update Zone Price	{
				price:integer
				}

Notes:

- **Create Station:**
  Admin can create new normal stations not connected to any station
- **Update Station:**
  Admin can update stations name
- **Delete station:**
  Admin can delete start, middle, end, transfer station.
  So, you have to handle each case, and create new routes named for example ‘new’ to substitute the deleted routes and the admin can update their names.
  create new records in the stationRoutes table.

- **Create Route:**
  Admin can create a route for the newly created station by choosing the station to connect to it, so you have to handle only two positions (start or end)
  We won't handle creating stations in the middle of routes
- **Update Route:**
  Admin can update routes name
- **Delete Route:**
  Admin can delete any route at the start or at the end of the graph
  either in forward ➡️ or backward direction ⬅️.
  So, if both directions are deleted, the status of the unconnected station will change and the position of the second station will change.
- **Handle requests:**
  You need to handle cases of acceptance and rejection of any request
- **Purchase Subscriptions through online payment:**
  When purchasing a subscription you will get a specific amount of tickets to use them when purchasing a ticket
  So, annual will provide 100 tickets, quarterly with 50 tickets, and monthly with 10 tickets
- **Purchase tickets through Subscriptions:**
  If the user has a subscription, she/he can purchase a ticket through it, the system should indicate the full ticket price, route, and transfer stations, then the user will have an upcoming ride on the date of the ticket
- **Purchase tickets through online payment:**
  If the user does not have a subscription, she/he can pay it through online payment, the system should indicate the full ticket price, route, and transfer stations, then the user will have an upcoming ride on the date of the ticket
- **Check Price:**
  Users can check the price of the ticket by specifying the origin and destination.
  So, you have to figure a way through the three tables(stations, routes, stationRoutes)
  Hint visited stations array
- **Simulate Ride:**
  Users can start a ride by choosing origin and destination from drop down menu and trip date so, the ride will be completed
- **Refund ticket:**
  Users can only refund the future dated tickets. So any upcoming ride with ticket will be deleted, also you have to handle the two ways of refunding tickets (pay by subscription or online payment)
  No need for a wallet, only transaction table is enough
  status will be pending

- req(6,7,8) will be implemented in the frontend

3. Wireframes