**Integrated Assignment**

**Developing a Server Side application using Node & Express**

**Instructions to use the project file provided:**

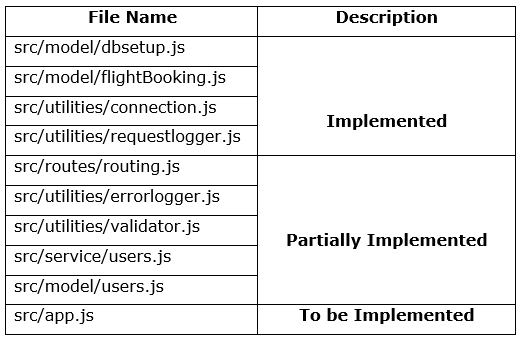
* Read the problem statement, examples and the other details provided carefully and implement the solution
* Download the project **FBWebService** in to your system and unzip it
* Install **node\_modules** using the command **npm install** and **start** your server
* Once the server started successfully, hit the **URL** **http://localhost:1050/setupDb** **(GET request)** to setup your database with initial set of documents
* Run index.html inside **parserModule** in **live server** to verify your code. Your application should not have any run-time/compilation errors for successful verification
* **DO NOT** alter the function name or the argument list of the function that is provided to you
* **DO NOT** add any new functions apart from the one given in the file where you write the solution
* **DO NOT** write codes that result in infinite loops/infinite recursive calls, because it just won’t work!

**Problem Description:**

**Infy Airlines** wants to automate the process of flight booking and management for which they want the implementation of the following.

* Book tickets in a flight
* View all bookings for any flight
* View all bookings done for a particular flight
* View all bookings done by a particular customer
* Update the booking of a customer

**In FBWebService Folder:**



**model/flightBooking.js: (Implemented)**

* This file contains a Class **FlightBooking**, which converts a generic object to **Flight Booking** object

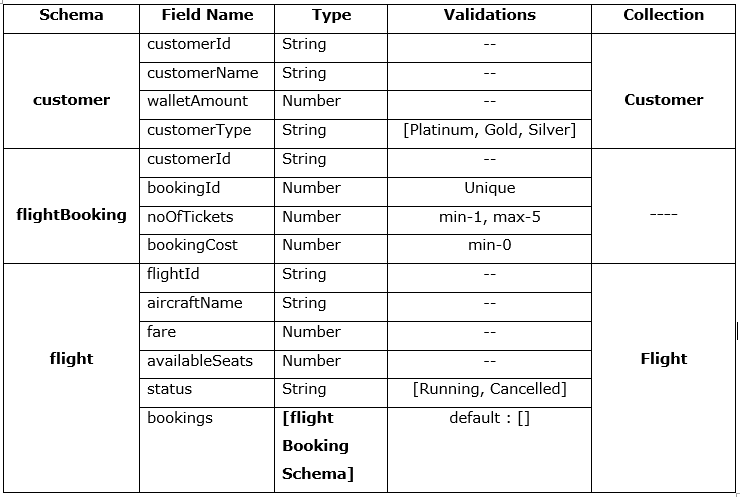
**utilities/requestlogger.js (Implemented)**

This file consists of a middleware function which logs all the requests made by the users into a text file **RequestLogger.txt.**

**utilities/connection.js (Implemented)**

* This file contains the database schemas and a connection object
* The connection object has the following two methods to establish connection with the database:
* **getCustomerCollection() –** gives model object of Customer collection
* **getFlightCollection() –** gives model object of Flight collection

The database consists of the following three schema’s



**model/dbsetup.js: (Implemented)**

This file contains the initial set of data for the collection and a method to insert the data into the respective collection in database.

**utilities/errorlogger.js: (To be implemented)**

* This file contains a function **errorLogger** which should log the full stack trace of the error that may get thrown during execution of the program
* If there is some error in the code the entire error stack should be appended to the **ErrorLogger.txt** file with the timestamp of when the error had occurred
* If there is any error in appending the error to the **Errorlogger.txt**, it should display the message **“Failed in logging error”** in console
* If error object’s status property is set, then the **response status** should be setto error objects **status** value
* Else, the **response status** should be setto **500** and the error message should be sent as a **JSON** in the given format **{“message” :<<message>>}**
* **errorLogger** should be exported as a module

**utilities/validator.js: (To be implemented)**

This file contains a Validator object with two methods i.e. **validateFlightId** and **validateBookingId**

* **validateFlightId**
* This method should accept **flightId** as a parameter and validate it
* It should check if the **flightId** matches the following pattern:

**<part1>>-<<part2>>**

**part1** -> **IND**

**part2**->three digits (where first digit cannot be 0)

**e.g. IND-101**

* If the validation fails, it should throw an error with message **“Error in flight Id”** after setting error status as **406**
* **validateBookingId**
  + This method should accept **bookingId** as a parameter and validate it
  + It should check that the bookingId should always be of 4 digits only
  + If the validation fails, it should throw an error with message **“Error in booking Id”** after setting the error status as **406**

**model/users.js: (partially implemented)**

* Instance of **connection** module is created by importing it
* This file contains **flightBookingDb** object with the following methods:
  + generateId()
  + checkCustomer()
  + checkBooking()
  + checkAvailability()
  + bookFlight()
  + getAllbookings()
  + customerBookingsByFlight()
  + getbookingsByFlightId()
  + updateCustomerWallet()

**Note**: Details of the mentioned methods are given in the next section

* Finally, **flightBookingDb** object is exported as a module

**flightBookingDb.generateId() :(Implemented)**

* + This method generates unique **booking Ids** for each new document that gets inserted
  + The generated **bookingId** is returned wrapped inside a promise

**flightBookingDb.checkCustomer() :(to be implemented)**

* This method should take **customerId** as a parameter and check whether customer with given customerId exists or not
* A connection should be established to the database by invoking the **getCustomerCollection** method of the **connection** object
* It should fetch the customer details from **Customer** collection for the given customerId
* If the customer detail exists, it should return the customer object
* Else, it should return **null**

**flightBookingDb.checkBooking() :(to be implemented)**

* This method should take **bookingId** as a parameter and check whether booking with given bookingId exists or not
* A connection should be established to the database by invoking the **getFlightCollection** method of the connection object
* It should fetch the Flight details from **Flight** collection for the given bookingId
* If the bookings exist for the given bookingId, return the corresponding flight object in which the booking is done
* Else, it should return **null**

**FlightBookingDb.checkAvailability() :(to be implemented)**

* This method should take **flightId** as a parameter and check whether flight with given flightId exists or not
* A connection should be established to the database by invoking the **getFlightCollection()** method of the connection object
* It should fetch the flight detail from **Flight** collection for the given flightId
* If a flight object for the given flightId is found, it should return the flight object
* Else, it should return **null**

**flightBookingDb.updateCustomerWallet(): (to be implemented)**

* This method should accept **customerId** and **bookingCost** as parameters and update the wallet amount of the customer
* A connection should be established to the database by invoking the **getCustomerCollection()** method of the connection object
* It should update the wallet amount for the given **customerId** by decreasing the **bookingCost**
* On successful updation, return **true**
* Else, return **false**

**flightBookingDb.bookFlight() :(to be implemented)**

* This method should accept **flightBooking** Object as a parameter, book tickets for the customer, update the number of seats available for the flight and the walletAmount of the customer
* A connection should be established to the database by invoking the **getFlightCollection** method of the connection object
* It should invoke **generateId** method of **flightBookingDb** to generate unique booking Id
* A new document should be added to the **bookings** of Flight collection with the values of **customerId, bookingId, noOfTickets, bookingCost**
* If new document is added successfully, update the number of seats available in the **Flight** Collection for the corresponding **flightId**
  + It should also check for successful updation of seats available in the **Flight** collection
  + If successfully updated, it should update the corresponding customer’s wallet amount by invoking **updateCustomerWallet** method that takes customerId and **bookingCost** as parameters
  + If the wallet amount is updated for the customer, it should return the **bookingId**. Else, it should throw an error with message **“Wallet not updated”** after setting the error status as **502 (Bad gateway)**
  + Else, if the seats available for the flight does not get updated, it should throw an error with message **“Seats not updated”** after setting the error status as **502 (Bad gateway)**
* Else, it should throw an error with message **“Booking failed”** with status **500**

**flightBookingDb.getAllBookings() :(to be implemented)**

* This method should not take any parameter and return all the bookings made in all the flights
* A connection should be established to the database by invoking the **getFlightCollection** method of the connection object
* It should fetch all bookings from all the flights.
* It should return an array of booking objects
* If no bookings are found, return **null**

**flightBookingDb.** **customerBookingsByFlight () :(to be implemented)**

* This method should take **customerId** and **flightId** as a parameter and return an array of all the bookings made by the customer in the given flight
* A connection should be established to the database by invoking the **getFlightCollection** method of the connection object
* It should fetch all the bookings made by a customer for the given flightId
* If bookings are found, it should return the same array of **bookings** object
* Else, it should return **null**

**flightBookingDb.** **getbookingsByFlightId() :(to be implemented)**

* This method should take flightId as a parameter and return all the bookings made in that flight
* A connection should be established to the database by invoking the **getFlightCollection()** method of the connection object
* It should fetch only the bookings detail from the flight collection for the given flightId.
* If bookings are found, it should return the same **bookings** array
* Else it should return **null**

**flightBookingDb.updateBooking() :(to be implemented)**

* This method should take **bookingId** and **noOfTickets** as parameters and update the booking details for the customer and do the other necessary changes in the customer and flight collection
* A connection should be established to the database by invoking the **getFlightCollection** method of the connection object
* It should update the noOfTickets for the given bookingId by adding the given noOfTickets, also reduce the available seats accordingly
* Also update the **bookingCost** for the booking by adding the required fare for the given noOfTickets **(bookingCost = fare\*noOfTickets)**

**Note:** Get fare from the flight object

* + If **bookingCost** is successfully updated, it should update the **wallet** **amount** by subtracting the booking cost for the customer corresponding to the booking
  + If wallet amount gets updated it should return the updated flight details by invoking **checkAvailability** method of **flightBookingDb** object
  + Else, it should return **null**

**service/users.js (partially implemented)**

* Import all the required modules
* **fBookingService** object is created with the following methods:
  + bookFlight()
  + getAllBookings()
  + customerBookingsByFlight()
  + getBookingsByFlightId()
  + updateBookings()

**Note:** Details of above methods are given below

* Finally, it should export **fBookingService** as a module

**fBookingService.bookFlight() :(to be implemented)**

* This method should take **flightBooking** object as a parameter and book the mentioned number of tickets for an existing Flight.
* It should validate the flightId by invoking appropriate **validateFlightId** method of Validator.js
* It should invoke the **checkCustomer** method of **flightBookingDb** objectby passing the customerId entered by the customer as a parameter which in turn returns customer details from the Customer collection
* If returned value is **null**, it should throw an error with message **“Customer not registered. Register to proceed”** and status as **404**
* Else, it should invoke **checkAvailability** method of **flightBookingDb** object by passing the flightId entered by the user as a parameter which in turn returns flight details from the Flight collection
* If the returned value is **null**, it should throw an error with message **“Flight Unavailable”** after setting error status **404**
* If the status of the returned flight is **“Cancelled”,** it should throw an error with message **“Sorry for the Inconvinience... *<<flightId>>* is cancelled!!”** after setting error status **400**
* If the **availableSeats** of the returned flight is **0,** it should throw an error with message **“Flight *<<flightId>>* is already full!!”** after setting appropriate error status
* If the **availableSeats** of the returned flight isless than the required **nOOfTickets,** it should throw an error with message **“Flight almost Full... Only <<availableSeats>> left!!”** after setting error status as **406**
* Else, it should assign the bookingCost value to the **FlightBooking** object **(bookingCost = noOfTickets \* fare)**
* It should then check if the available wallet amount is sufficient for booking or not.
* If not, it should throw an error with message **“Insufficient Wallet Amount. Add more Rs. <<amountNeeded>> to continue booking”** after setting error status as **406**
* Else, it should invoke the **bookFlight** method of **flightBookingDb** object which in turn returns the bookingId
* Finally, it should return the same bookingId

**fBookingService.getAllBookings(): (to be implemented)**

* This method should not take any parameter and return all the bookings over all the flights
* It should invoke the **getAllBookings()** method of **flightBookingDb** objectwhich in turn returns the array of booking details of all the flights
* If the returned value is **null,** it should throw an error with message **“No Bookings is found in any flight”** after setting error status as **404**
* Else, it should return the same array of bookings

**fBookingService.customerBookingsByFlight(): (to be implemented)**

* This method should take customerId and flightId and return the corresponding customer booking details in the given flight
* It should invoke the **checkCustomer** method of **flightBookingDb** object by passing customerId as parameter,which in turn returns the customer details
* If the returned value is null, it should throw an error with message **“Customer not found”** and status 404
* Else, it should invoke the **checkAvailability** method of **flightBookingDb** by passing flightId as parameter,which in turn returns the corresponding flight details
* If the returned value is null, it should throw an error with message **“Flight detail not found”**
* Else, it should invoke the **customerBookingsByFlight()** method of **flightBookingDb** by passing customerId and flightId as parameter which in turn returns the booking details for a customer in the given flightId.
* If the returned value is **null,** it should throw error with message ***“No Bookings found for <<customerId>> in <<flightId>>”*** after setting error status as **404**
* Else, it should return the same booking details array

**fBookingService.getbookingsByFlightId() :(to be implemented)**

* This method should take flightId as a parameter and returns all the bookings done in that flight
* It should invoke the **getbookingsByFlightId** of **flightBookingDb** by passing **flightId** which in turn returns the array of booking objects or null
* If the returned value is **null,** it should throw an error with message ***“No Bookings found in <<flightId>>”*** after setting error status as **404**
* Else, it should return the same booking details array

**fBookingService.updateBooking() :(to be implemented)**

* This method should take **bookingId** and **noOfTickets** and update the corresponding booking
* It should invoke the **checkBooking** method of **flightBookingDb** object by passing bookingId as parameter,which in turn returns the flight object or null
* If the returned value is **null**, it should throw an error with message ***“No Bookings with bookingId <<bookingId>>”*** after setting the appropriate error status
* If the status of returned flight is **“Cancelled”**, it should throw an error with message ***“Sorry for the Inconvenience... <<flightId>> has been cancelled!!”*** after setting the error status as **406**
* If the availableSeats of returned flight is **0,** it should throw an error with message ***“Flight is already Full. Can't Book more tickets”,*** after setting the error status as **406**
* If the availableSeats of returned flight is less than the required number of tickets**,** it should throw an error with message ***“Flight almost Full. Only <<availableSeats>> seat left”,*** after setting the error status as **406**
* Else, it should check if the customer has sufficient walletAmount to book the required number of tickets by invoking **checkCustomer** method of flightBookingDb object
* If wallet amount is less, it should throw an error with message ***“Insufficient Wallet Amount. Add more Rs. <<amountNeeded>> to continue booking”*** after setting error status as **406**
* Else, it should invoke **updateBooking()** method of **flightBookingDb** object by passing **bookingId** and **noOfTickets** which in turn returns the updated flight details, return the same
* If the returned value is **null**, it should throw an error with message ***“update failed”*** after setting error status as **502**

**routing.js: (to be implemented)**

* Import all required modules
* The following URI’s should be configured for booking, fetching details & updating bookings respectively

**URI:- /bookFlight**

* Configure the instance of Router, to handle the **POST** request for the given URI.
* Once the request is received, it should create and populate **FlightBooking** object with the values present inside the request body
* It should invoke **bookFlight** method of **fBookingService** by passing the **FlightBooking** object, which in turn returns bookingId wrapped inside a promise
* If the promise is successful it should populate the JSON response in the given format: ***{"message": "Flight booking is successful with booking Id: <<bookingId>>”}*** with status as **201**
* Else if the promise fails, it should forward the control to next handler by passing the error object as parameter

**URI:- /getAllBookings**

Configure the instance of Router, in order to handle the **GET** request for the given URI.

* It should invoke **getAllBookings** method of **fBookingService**, which in turn returns an array of booking objects
* If the promise is successful it should populate the JSON response with the bookings returned by the **getAllBookings** of **fBookingService**
* Else if the promise is failed it should forward the control to next handler by passing the error object as parameter

**URI:- /customerBookings/:customerId/:flightId**

It should also configure the instance of Router, to handle the **GET** request for the given URI

* It should invoke **customerBookingsByFlight** of **fBookingService** objectby passing the **customerId** and **flightId** obtained from the **URL**, which in turn returns an array of bookings done by the customer for the given flightId
* If the promise is successful it should populate the JSON response with the bookings returned by the **customerBookingsByFlight** of **fBookingService**
* Else if the promise is failed it should forward the control to next handler by passing the error object as parameter

**URI: /bookingsByFlight/:flightId**

It should also configure the instance of Router, to handle the **GET** request for the given URI

* It should invoke **getbookingsByFlightId** of **fBookingService** objectby passing the **flightId**, which in turn returns array of bookings done for that flight
* If the promise is successful it should populate the JSON response with the bookings returned by the **getbookingsByFlightId** of **fBookingService**
* Else if the promise is failed it should forward the control to next handler by passing the error object as parameter

**URI: /updateBooking/:bookingId**

It should also configure the instance of Router, in order to handle the **PUT** request for the given URI

* It should invoke **updateBooking()** of **fBookingService** objectby passing the **bookingId** obtained from the **URL**  and **noOfTickets** from the request body, which in turn returns an flight object
* If the promise is successful it should populate the JSON response in the given format: ***{"message": "Booking successfully updated!! updated flight details <<flight>>”}***
* Else if the promise fails, it should forward the control to next handler by passing the error object as parameter

**app.js: (To be implemented)**

* All the required modules should be imported
* Middleware’s should be organized properly for their appropriate functionality
* The application should listen to requests coming through port **1050**

**~~~~~~~~~~~~~~~~~~All The Best~~~~~~~~~~~~~~~~~**