To achieve the given functionality provided in the Business Scenario for the airport department, I have done the following changes.

**Renaming File names from NodeJS05 project:**

Filename ‘routes/category.js’ is changed to ‘routes/airport.js’.

* Filename ‘model/category.js’ is changed to ‘models/airport.js’.
* Filename ‘controller/category.js’ is changed to ‘models/Airport.js’.
* Filename ‘views/pages/category.pug’ is changed to ‘views/pages/airports.pug’.
* Filename ‘views/pages/category.html’ is changed to ‘views/pages/airports.html’.

**APP.JS**

Various libraries have been imported

* **import** *express* **from 'express'**;

ExpressJS is a web application framework for NodeJS which is available in NPM registry and can be downloaded using CLI of NPM. To install something from NPM registry we can use the command NPM install and to test the functionality we can use NPM test command.

* **import mongoose from "mongoose"**;

Mongoose is an Object Data Modelling (ODM) library for MongoDB and NodeJS. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB. The combination of ODM database and ODM is well known in the community of Node because the storage of documents and the queries look like JSON which is familiar to JavaScript developers. It is one of the powerful query languages as it uses JSON instead of concatenating strings.

* **import** bodyParser **from 'body-parser'**;

Body-parser extract the entire body portion of an incoming HTTP request stream and exposes it on req. body from within routes and use that data to perform some manipulations in the database.

This extraction is necessary when we want to know the information contained in Post, Put or Patch HTTP requests whose bodies contains the main information. The middleware was a part of Express.js earlier but now we need to install it separately. This body-parser module parses the JSON, buffer, string and URL encoded data submitted using HTTP POST request. URL encoded and json functions of body parser have also been used in this project. Using URL encoded a middleware is returned which parses only specific type of bodies and makes sure that type option defined matches the content type header of request.

After this I have changed the name of the prefix to AIRPORT\_PREFIX and the categoriesRouter to ariportsRouter which are imported from the airports.js file present inside the routes folder denoted by the statement.

**import *airportsRouter***, { ***AIRPORT\_PREFIX*** } **from './routes/airports'**;

Further, I have changed the database name from Toyshop to ‘AirportDepartment’. So, it will create a new database if there is no existing one with the name ‘AirportDepartment’ or else will use the existing one if there is any already available.Then, I have passed AIRPORT\_PREFIX and airportsRouter as a parameter in app.use() which is called every time a request is sent to the server. **use() is used to mount to a specified path and the corresponding function is executed when the path matches.**

**Models/airport.js**

I have created a new schema called ‘airportSchema’ using mongoose.schema() method present inside the mongoose library and created 2 new columns as follows:

**const** airportSchema = **new mongoose**.**Schema**({  
 **Airport\_Code**: {  
 **type**: ***String***,  
 **required**: **true**,  
 **length**: 3  
 },  
 **Phone\_Number**: {  
 **type**: ***Number***,  
 **required**: **true**,  
 **length**:10  
 },  
  
});

I have changed the datatype for Phone\_Number from ‘String’ to ‘Number’ type so that it does not accept characters as input.

I have stored this mongoose collection in a constant called ‘Airport’ and exported it so that it can be used in other files.

**const *Airport*** = **mongoose**.model(**'Airport'**, airportSchema, **'Airport'**);  
  
**export default *Airport***

mongoose.model method creates a new mongoDB model using the specified parameters.

**Controller/Airport.js** ,

I have imported the collection Airport from the files models/airtport.js.

I have created a constant variable called AIRPORT\_PREFIX and assigned it the value **‘/airports’** which is exported from controller/Airport.js and imported by routed/airports.js. So, this makes it mandatory to enter /airports in the URL. Further, the controller class AirportController extends the BaseController and super() keyword is used to refer the constructor of the BaseController. I have passed ‘Airport’ and ‘AIRPORT\_PREFIX’ as a parameter to super() which are imported from ‘..models/airport’ and created above respectively.

**class** AirportController **extends** BaseController {  
 constructor() {  
 **super**(***Airport***, ***AIRPORT\_PREFIX***);  
  
 **this**.getForms = **this**.getForms.bind(**this**);  
 **this**.validate = **this**.validate.bind(**this**);  
 }

getForms() will render the form from the file ‘pages/airports.html’.

getAll() will use mongoose method called ‘find()’ to return all the airport departments data from the schema in the JSON(JAVASCRIPT OBJECT NOTATION) format. Update() will use mongoose method called ‘findByIdAndUpadte()’ which takes id as a parameter from HTTP request body, updates the data and retrieves the updated information in the JSON format.

**BASE.JS**

Create() will use mongoose method ‘create()’ to enter new AirportDepartment data in the database and will return that information in the json format. Delete() will use mongoose method ‘findByIdAndDelete()’ which will take id as a parameter, deletes the particular record matching with id and returns the ‘success’ message

**Views**

layout.pug file will provide the basic layout with header and footer to display the form.airports.html provides the form with two fields Airport\_Code and Phone\_Number where user can enter the data. In airports.pug file, first I have used ‘each’ loop to display all the AirportCode and PhoneNumber. In airports.pug file , forms are created for CREATE,UPDATE and DELETE functionalities with label and textboxes and submit button.

All the style properties for these forms are inherited from the public/bootstrap.min.cs and style.css

**Validation Rules**

Various validations has been applied according to the business rules .

For field ‘Airport\_Code’, 4 business rules have been applied which will check if the Airport\_Code is null or not , it is of string type or not and the length of the AirportCode must be 3 characters.

For the field ‘PhoneNumber’, 4 business rules have been applied which will check if the PhoneNumber is null or not , it is in numeric format or not and the length of the PhoneNumber is exactly 10 characters.Here, trim() removes the leading and trailing whitespaces

validate() {  
 **return** [  
 check(**'Airport\_Code'**)  
 .exists().withMessage(**'Field Airport\_Code must exist'**)  
 .isString().withMessage(**'Airport Code must be of the string type'**)  
 .trim().isLength({ **min**: 3}).withMessage(**'Airport Code must be at least 3 characters long'**)  
 .trim().isLength({ **max**: 3}).withMessage(**'Airport Code must be at least 3 characters long'**),  
 check(**'Phone\_Number'**)  
 .exists().withMessage(**'Field Phone\_Number must exist'**)  
 .isNumeric().withMessage(**'Phone Number must be of the numeric type'**)  
 .isLength({ **min**: 10}).withMessage(**'Phone number must be exactly 10 characters long'**)  
 .isLength({ **max**: 10}).withMessage(**'Phone number must be exactly 10 characters long'**)  
 ]  
}