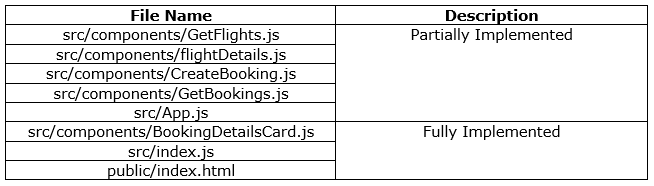
**Integrated Assignment**

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

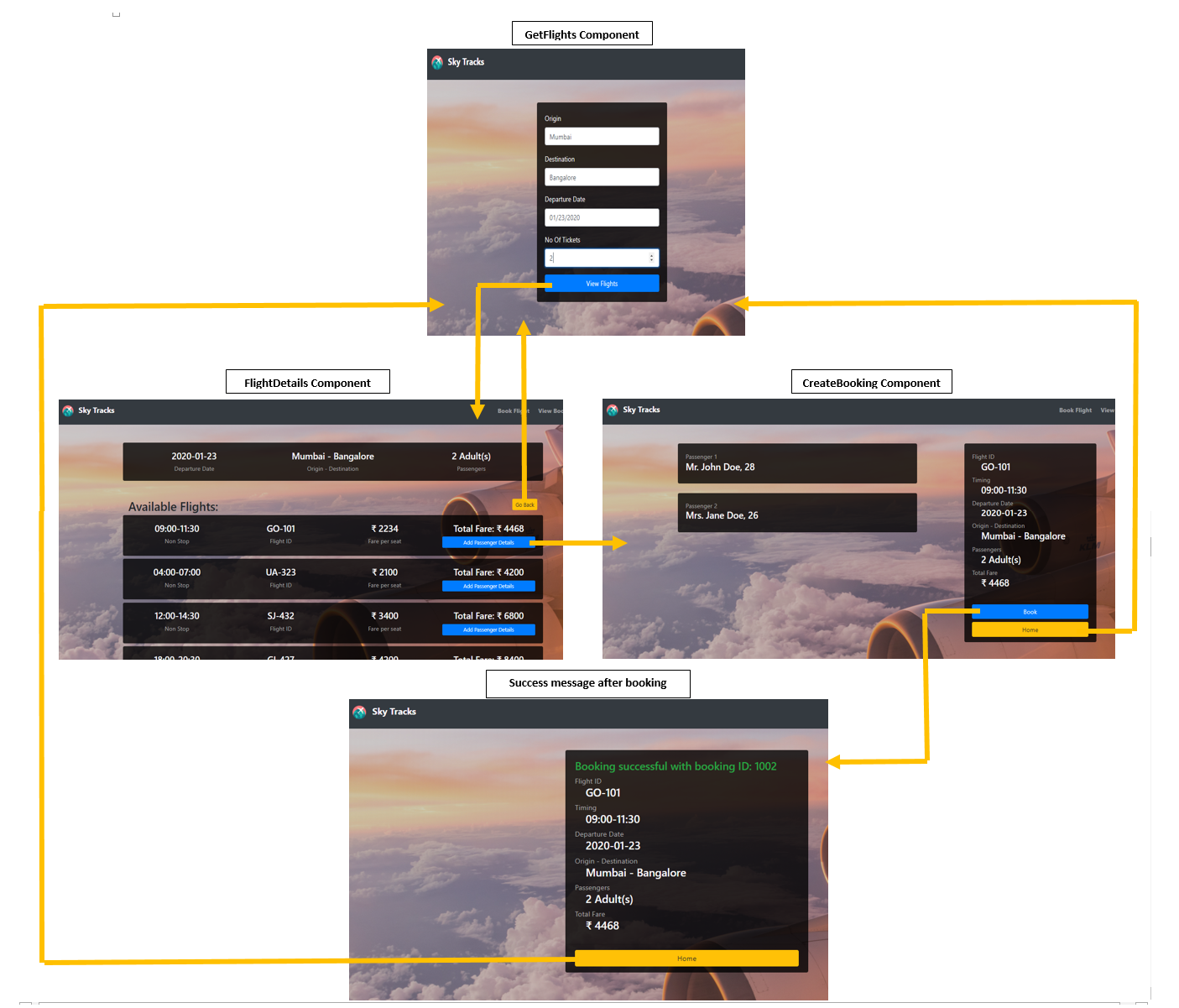
* You need to code the assessment in the folder containing partially coded project
* Read the problem statement, examples and the other details provided carefully and implement the solution
* Download the project **SkyTracks-toTrainee** in to your system and unzip it
* The unzipped folder should contain two sub folders i.e. **SkyTracksUI** and **SkyTracksWS**
* **SkyTracksUI** should contain the client side code
* **SkyTracksWS** contains the implemented server side code
* **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!
* Execute **npm install** command in **SkyTracksUI** and **SkyTracksWS** folder
* Execute **node app** command in **SkyTracksWS/src** to start the **web service (running on PORT 1050)**
* **USE APPROPRIATE BOOTSTRAP CLASSES WHERE EVER NECESSARY**
* **DO NOT add, remove or alter any state in any of the components.**

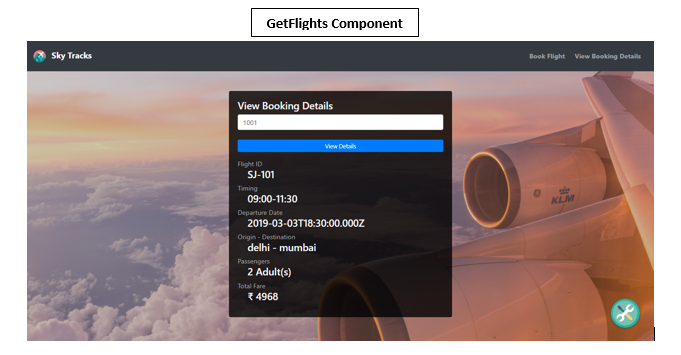
**SkyTracks** is a website which allows the user to book flight tickets. Their website allows the user to check available flights between two places, book the flight as well as view flight booking details.

**Implementation details of SkyTracksUI folder:**



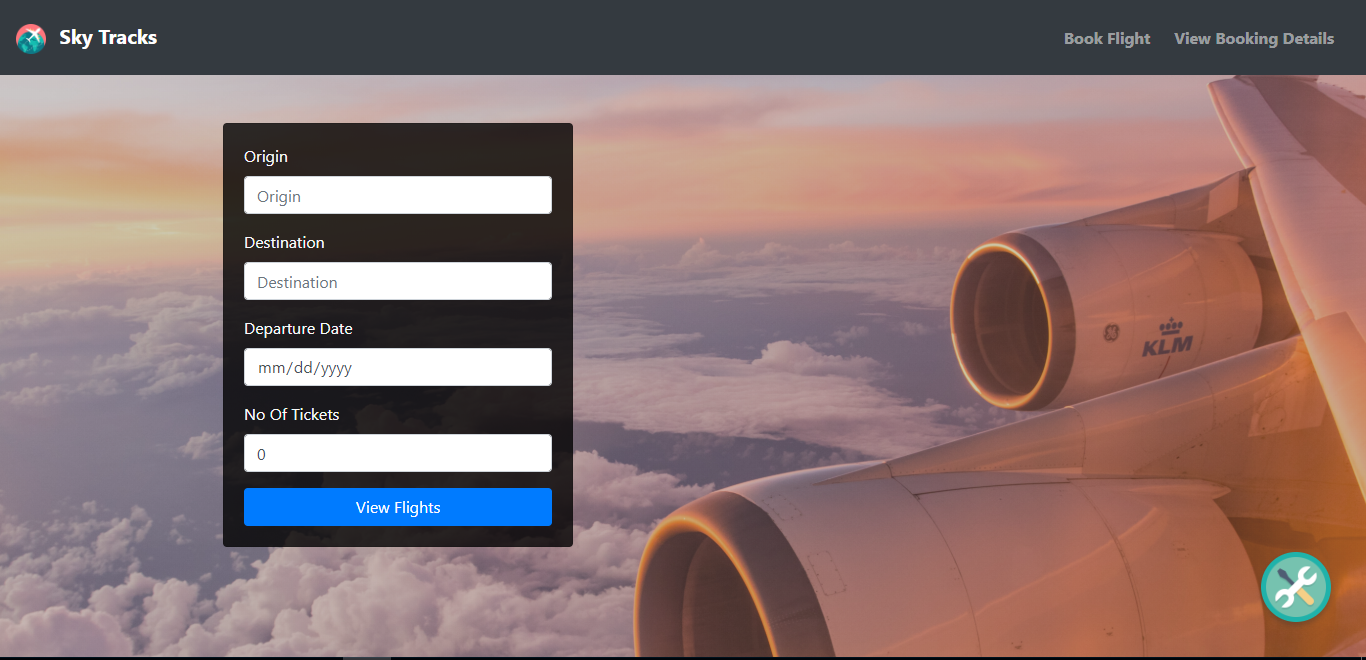
**A complete flow of the application**

****



**src/App.js**

* Configure the routes for **/bookFlight** that loads **GetFlights** Component and **/viewBooking** that loads **GetBooking** component.
* Configure another route to redirect to **GetFlights** component in case of incorrect route path.
* When the application loads we must have the following view:



**src/components/GetFlights.js**

All the **states** for the **GetFlights** component are defined as given:

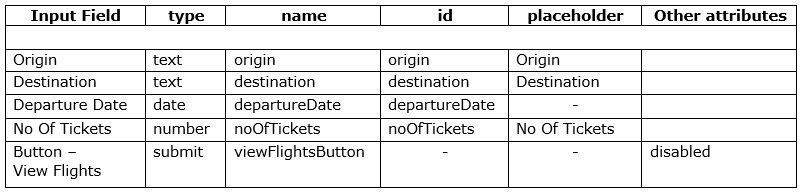
* **availableFlights:** Stores all the available flights between two given places.
* **form:** Stores each of the input given by the user in the form.
* **formErrorMessage:** It should store error message for each input field when validations fail.
* **formValid:** It should store the validity of each input field.
* **errorMessage:** It should store the error messages from axios request.

The **methods** in **GetFlights** component are given below:

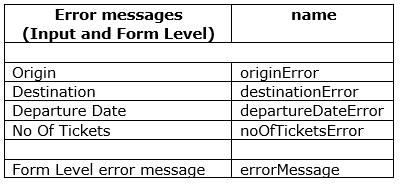
* **render()**
  + If the availableFlights state is not null, then **FlightDetails** component is rendered with **form** object and **availableFlights** in state as props with props name: flightData and availbaleFlights respectively.

Ex: If A is a component with “sampleData” props with props name “details”, then it is rendered as <A details={sampleData}></A>

* + Else, the form should be rendered inside a card (whose structure already given) as shown in the above snapshot.
  + The form has 4 inputs, whose details are mentioned as below:



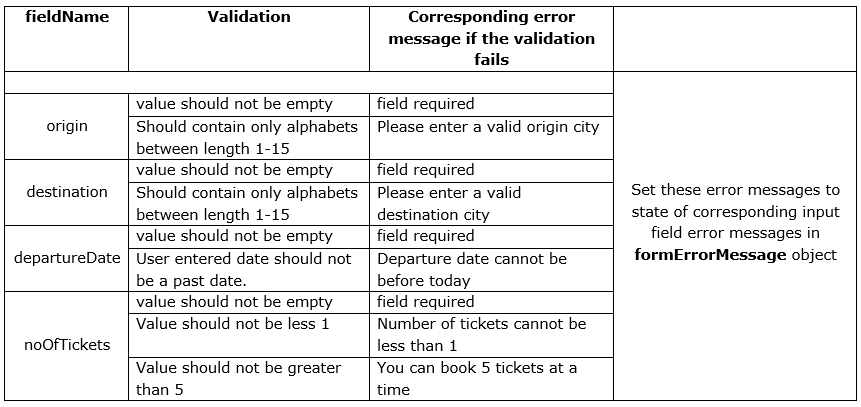
* + The **value attribute** for each input field should be bind with the corresponding state of form object.
  + The error messages for each input field and the form level error message should be displayed in a **span tag** with the following name attributes.
  + Form level error messageshould be displayed below the button in the form, whereas the input field error messages should be displayed below their respective input fields.



* **handleChange(event)**
  + This method takes event as a parameter and sets the value entered by the user in each input to the corresponding state of **form** object.
  + After setting the state value, it should invoke **validateField()** method with the **name** of the input and its **corresponding value** as parameters.
  + This method should be invoked whenever there is any change in value of the input fields.

**Note:** Do not return any value from this method**.**

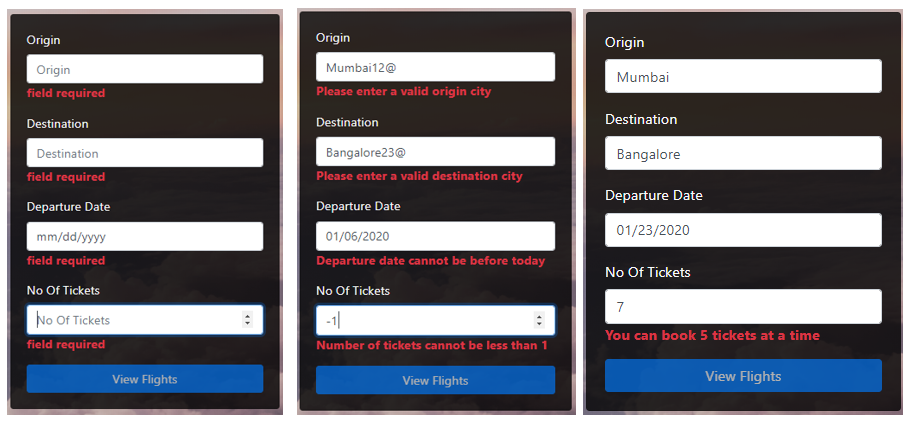
* **validateField(fieldName , value)**
  + This method should take name of the field and the corresponding value entered, to validate it based on the given requirements.
  + The validation for each input is as mentioned in the table given below:



* + Whenever the **validation is successful** for a particular input, the corresponding valid state in **formValid** object should be set to **true.**
  + Whenever the **validation fails** for a particular input, the corresponding valid state in **formValid** object should be set to **false**.
  + Only If all the validations for the input fields are successful, the **buttonActive** state in formValid state object should be set to true.
  + **Do not return** any value from this method.

**Note:** The **View Flights** button should be disabled until all the fields have valid input. The **buttonActive** state of **formValid** object should be used as per requirements to disable and enable the button.

**Sample snapshots for error messages:**



* **submitBooking()**
  + Set the **errorMessage** and **availableFlights** states with **empty string** value.
  + It should send an **AXIOS** **GET** request to the **URL http://localhost:1050/getFlights/<<origin>>/<<destination>>** where origin and destination are route parameters which are fetched from the form object in state.
  + If the request is successful, the response will have an Array of object which contains all the details of available flights. This response data should be set to the

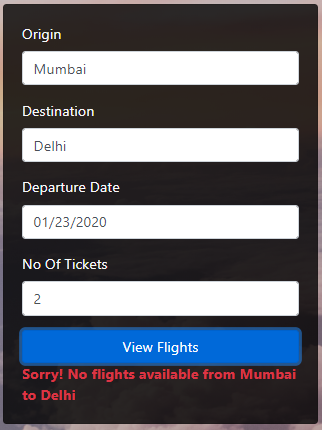
**availableFlights** state.

* + If the request fails, set **errorMessage** with message of the error response data and **availableFlights** to null

**Note:**

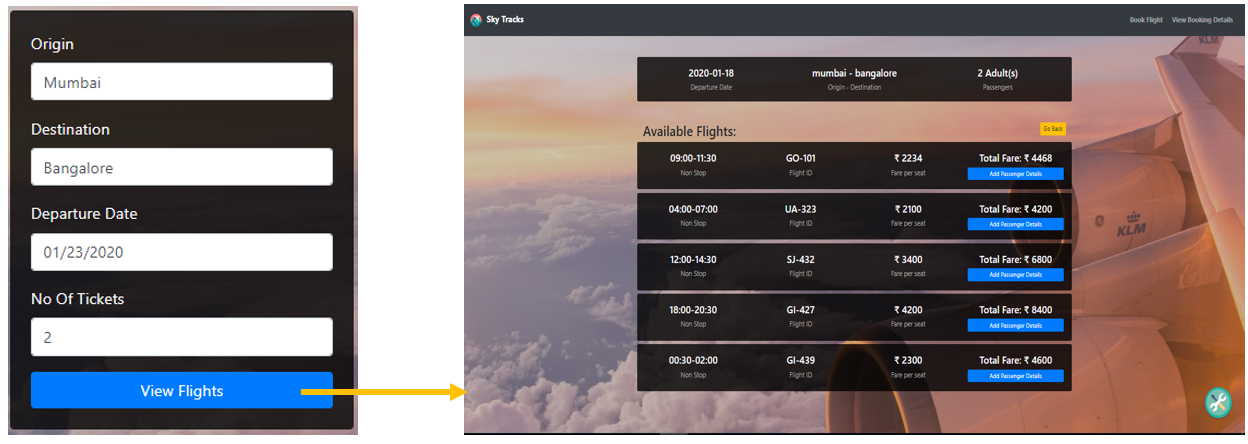
* + If the request is successful, the errorMessage (if any) should disappear and VICEVERSA
  + **Do not return any value from this method**

**Sample snapshot for error message:**



* **handleSubmit()**
  + This method takes event as a parameter and prevents the default behavior of form submission
  + Invoke the **submitBooking** method.
  + handleSubmit method should be invoked on submission of the form.

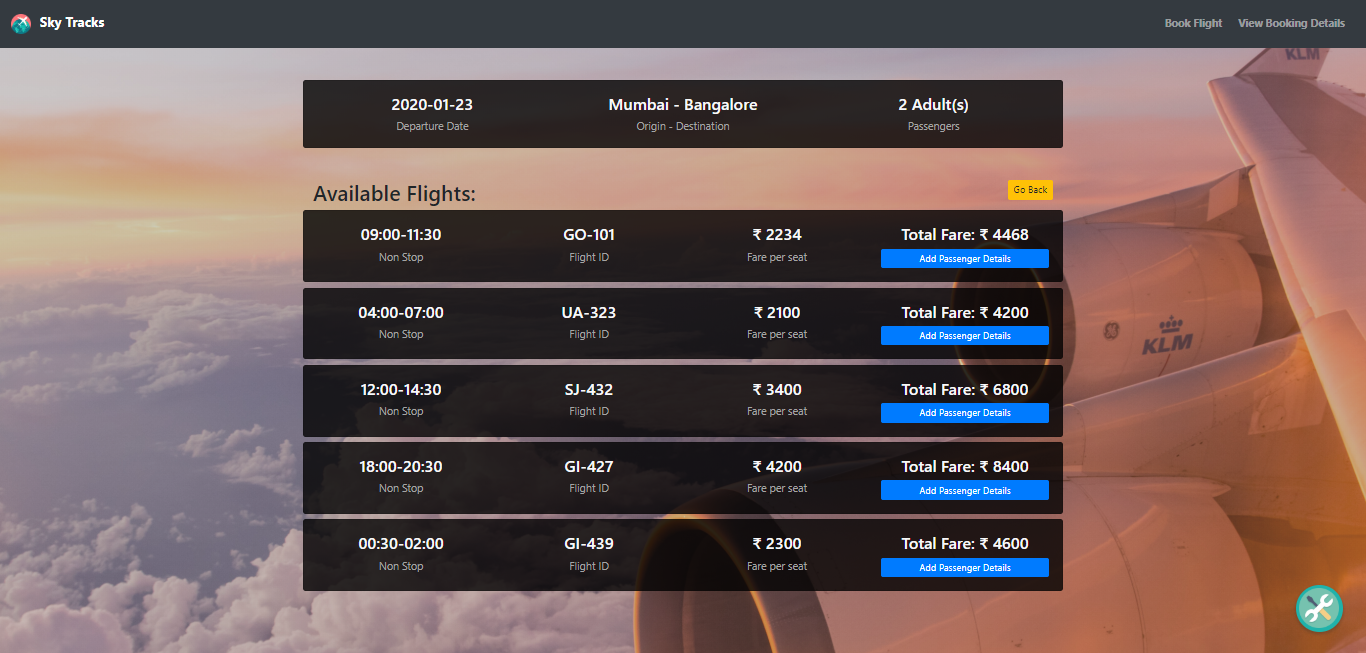
When the “**View Flights**” button is clicked, form will be submitted, **availableFlights** state will be set with axios request data and hence **FlightDetails** component is loaded.



**src/components/flightDetails.js**

On load of **FlightDetails** component, the view will be as shown below:

(For Inputs: Origin – Mumbai, Destination – Bangalore, Departure Date – 01/23/2019 , No Of Tickets – 2)



All the **states** for the **FlightDetails** component are defined as given:

* **flightData:** Stores the flightData props received from GetFlights component.
* **availableFlights:** Stores availableFlights props received from GetFlights component.
* **bookingDetails:** Stores all the booking details of a particular user. Its initial value is null

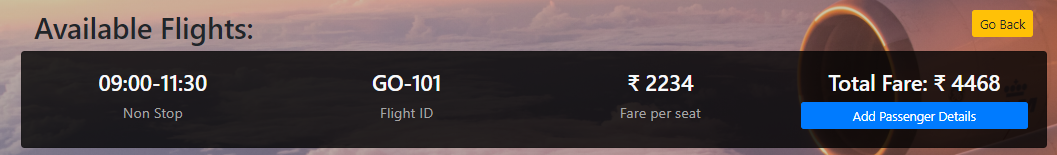
The **methods** in **FlightDetails** component are given below:

* **render()**
  + If **availableFlights** state is **null**, then **GetFlights** component should be rendered.
  + Else if **bookingDetails** state is **not null**, then CreateBooking component should be rendered with bookingDetails state as props with

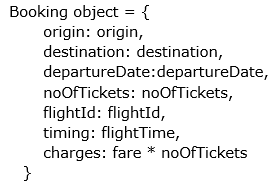
props name: bookingDetails

* + Else, cards should be rendered which displays the details of available flights as shown in snapshot above.
  + The card that displays Departure, Origin, Destinations and Passengers is already given. In the same way, create cards which displays the details of available flights by iterating over the availableFlights state.
  + The button “Add Passenger Details is small size block button. Its name should be “addPassenger”.
  + The button “Go Back” is small in size. Use appropriate bootstrap class for button color. Its name should be “goBack”.

Each card should be rendered as shown below:

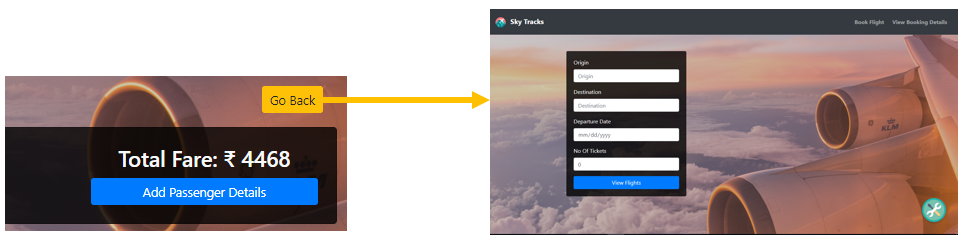


* **setBookingDetails(flightId,flightTime,fare) -** implemented
  + This method is invoked on click of the “Add Passenger Details” button.
  + It takes the **flightId , flightTime , fare** of the selected flight as parameters.
  + An object is created as –

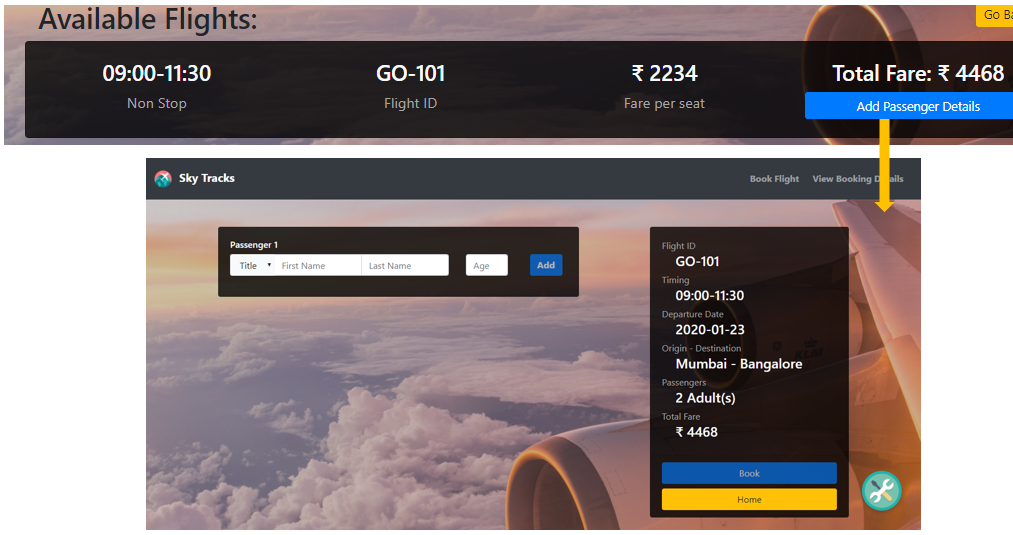


* + The bookingDetails state is set with the objected created.

On clicking the “Go Back” button, set the state of availableFlights state to null.



On click of the “Add Passenger Details”, bookingDetails state will be populated with data and CreateBooking component will be rendered whose view should be as shown below: (On selecting the flight with Flight Id – GO-101)



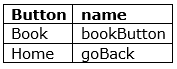
**src/components/CreateBooking.js**

All the **states** for the **CreateBooking** component are defined as given:

* **bookingDetails:** Stores the bookingDetails props supplied to this component.
* **passengerData:** Stores the details of each passenger in the form of an object in an array.
* **form:** Stores each of the input given by the user in the form.
* **formErrorMessage:** It should store error message for each input field when validations fail.
* **formValid:** It should store the validity of each input field.
* **errorMessage:** It should store the error messages from axios request.
* **successMessage:** Stores the success response from the axios request.
* **goBack:** Stores either true or false. This is used to go back to previous components.

The methods in **CreateBooking** component are as follows:

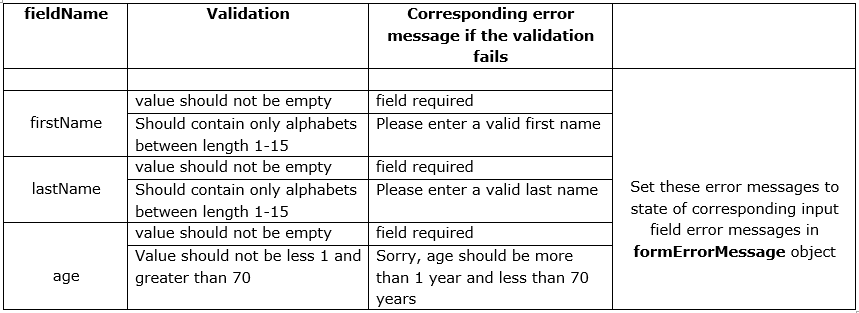
* **render()**
  + If the **goBack** state is true, render the **GetFlights** component.
  + If **successMessage** state is empty, then the view should be rendered as shown in the snapshot above by following the below given details.
* A partial code structure is already given.
* Inside the card body, render the **BookingDetailsCard** component (which is fully implemented) by passing **bookingDetails** state as props with
* **props name**: **bookingDetails**.
* For example: If **A** is a component, in which “someData” has to be sent as props with props name as “details”, then it would be rendered as
* **<A details={someData} />**
* Inside the card footer add the Book and Home buttons.



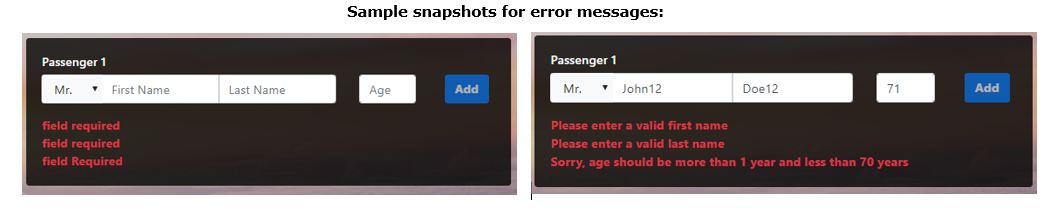
* The **Book** button is disabled till the number of passenger details (ie passengerData length) added is equal to the no of tickets booked (no of tickets in bookingDetails)
* On clicking the **Home** button, **goBack** state is set to **true**.
* Any axios error message should be displayed below these buttons.
  + Else, **displayBookingSuccess** method should be invoked.
* **handleChange(event)**
  + This method takes event as a parameter and sets the value entered by the user in each input to the corresponding state of **form** object.
  + After setting the state value, it should invoke **validateField()** method with the **name** of the input and its **corresponding value** as parameters.
  + This method should be invoked whenever there is any change in value of the input fields.

**Note:** Do not return any value from this method**.**

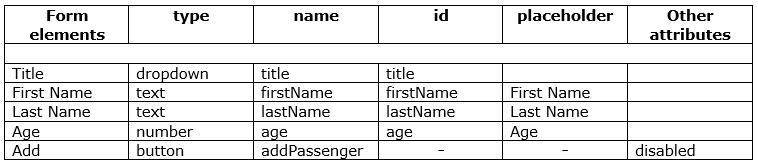
* **validateField(fieldName , value)**
  + This method should take name of the field and the corresponding value entered, to validate it based on the given requirements.
  + The validation for each input is as mentioned in the table given below:



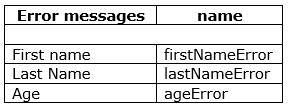
* + Whenever the **validation is successful** for a particular input, the corresponding valid state in **formValid** object should be set to **true.**
  + Whenever the **validation fails** for a particular input, the corresponding valid state in **formValid** object should be set to **false**.
  + Only If all the validations for the input fields are successful, the **buttonActive** state in formValid state object should be set to true.
  + **Do not return** any value from this method.



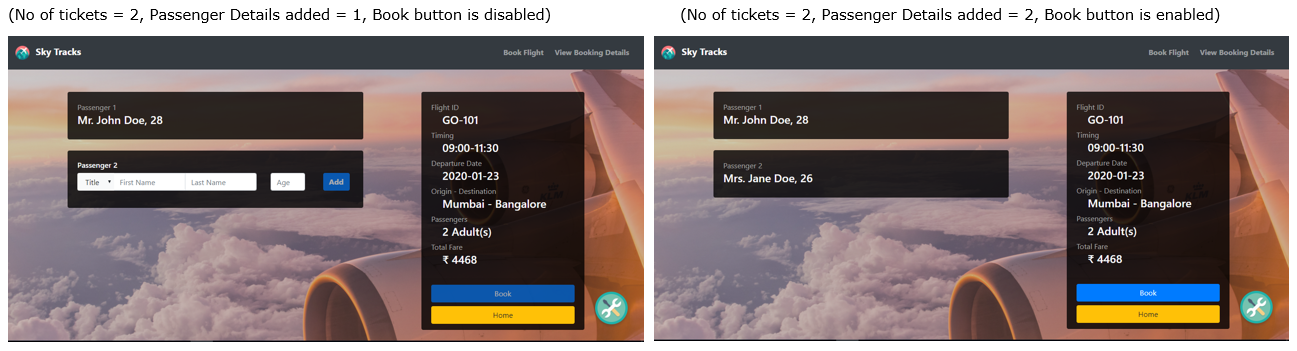
* **setPassengerData()**
  + Add the **form** object in state to the **passengerData** array in state.
  + Reset the **form** and the **formValid** object in state to its initial values.
* **getPassengerData()**
  + This method is use to take the passenger data (first name, last name , age) as input. A partial code structure is already given for the same.
  + The input fields are already present. Add the below given details to the same.



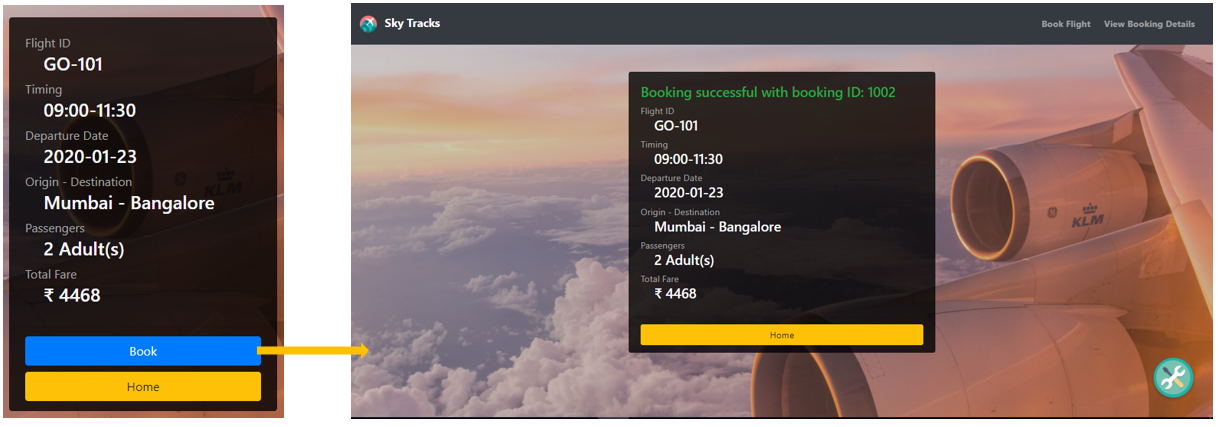
* + The **value attribute** for each input field should be bind with the corresponding state of form object.
  + All the error messages input fields should be displayed in a **div** tag, where “Display the formErrorMessages here” comment is written.

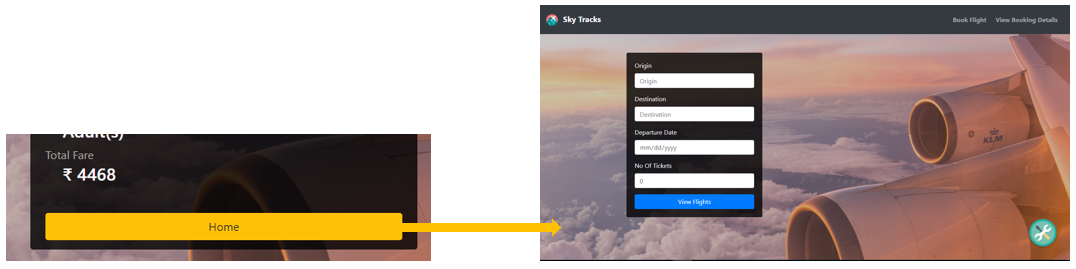


* **Add** button is disabled till all the inputs are valid.



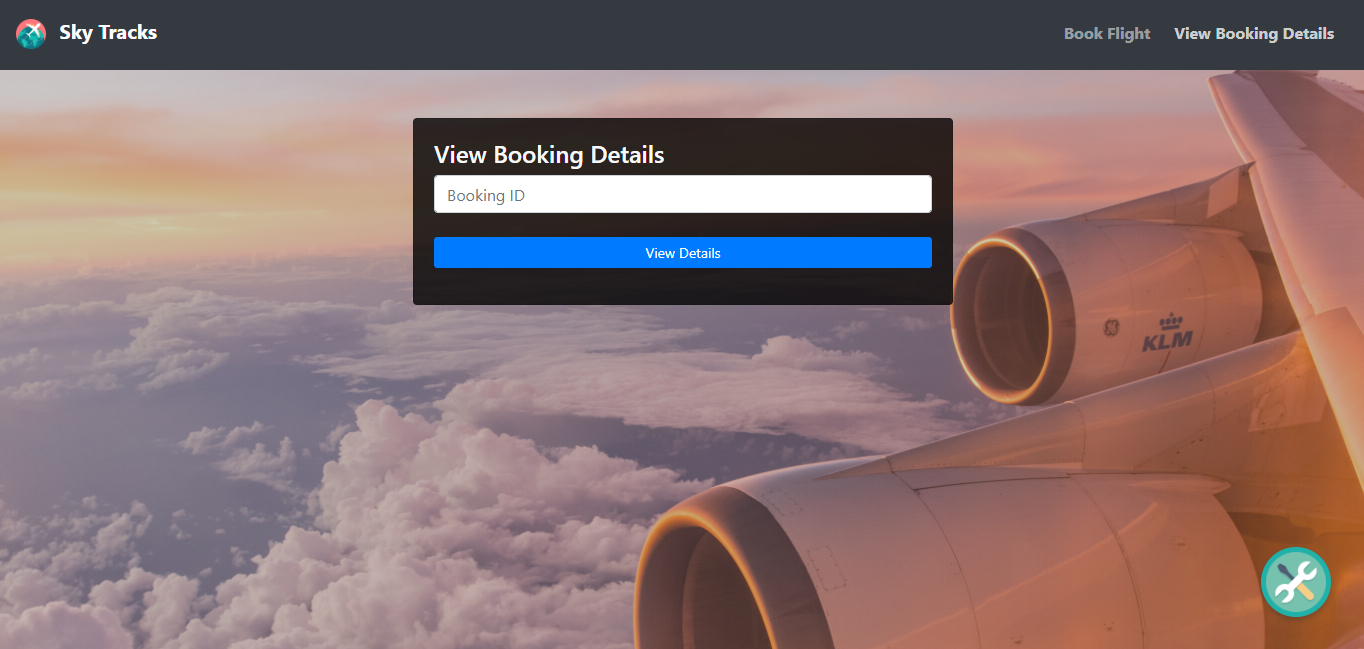
* **book()**
  + This method is invoked on click of the **Book** button.
  + Set the state of errorMessage and successMessage as “”.
  + Make an **AXIOS POST** request to the url "http://localhost:1050/bookFlight/" with **bookingData** (already populated) as the post data.
  + If the request is successful, set the successMessage state with the response data and errorMessage as “”.
  + If the request fails, set the errorMessage state to error message from the response and successMessage as “”.
* **displayBookingSuccess()**
  + This method is used to display the successMessage in the form of a card as shown below.
  + In the heading **h4,** add the **bookingId** from the successMessage**.**
  + Render the **BookingDetailsCard** component with bookingDetails object in state as props with props name: bookingDetails.
  + In the card footer, add a **Home** button (name: homeButton), on click of which **goBack** state is set to **true**.





* **src/components/GetBookings.js**

On clicking the View Booking Details in the Navbar, **GetBooking** component is rendered as shown below.

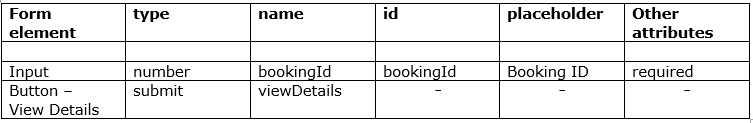


All the **states** for the **GetBooking** component are defined as given:

* **bookingData:** Stores the booking data fetched using axios request. Initial value is null.
* **bookingId:** Stores the booking id entered by the user in the form.
* **errorMessage:** It should store the error messages from axios request.

The methods in **GetBooking** component are as follows:

* **render()**
  + This method should render the form with an input field and a button.
  + The details are as given below:



* + The **value attribute** of input field should be bind with the **bookingId** in state.
  + The error message for the form should be displayed in a span tag (name: errorMessage) below the button.
  + If the bookingData state is not null, then render the BookingDetailsCard with bookingData object of state as props with props name: bookingDetails.
* **handleChange(event)** - implemented
  + This method takes event as a parameter and sets the value entered by the user in each input to the corresponding state of **form** object.
  + After setting the state value, it should invoke **validateField()** method with the **name** of the input and its **corresponding value** as parameters.
  + This method should be invoked whenever there is any change in value of the input fields.
* **handleSubmit()**
  + This method takes event as a parameter and prevents the default behavior of form submission
  + Invoke the **fetchBooking** method.
  + handleSubmit method should be invoked on submission of the form.
* **fetchBooking()**
  + Set the state of bookingData and errorMessage to null and “”.
  + Make an AXIOS GET request to the URL "http://localhost:1050/viewBookingDetails/<bookingId>" with **bookingId** as route parameter.
  + If the request is successful, then set the **bookingData** state with the response data and errorMessage to “”
  + Else, set the **errorMessage** state with the error response message and bookingData to null

On entering Booking Id: 1002 and clicking on View Details:

