OULU UNIVERSITY OF
APPLIED SCIENCES

Kasra Ariyaeimehr

FLIGHT LOGGING ENTRY FORM

Creating a form and results table for flight logging purposes

FLIGHT LOGGING ENTRY FORM

Creating a form and results table for flight logging purposes

Kasra Ariyaeimehr
Bachelor's thesis
Spring 2018
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author(s): Kasra Ariyaeimehr
Title of Bachelor´s thesis: Developing a web Application using ReactJS frame-work
Supervisor(s): Lasse Haverinen, Pekka Alaluukas
Term and year of completion: Number of pages:

The purpose of this thesis was to develop an interactive front-end for an already developed API, and already established development environment. The project was commissioned as a thesis project by the owner and moderator of cavokeapp.com which is an aviation related website for both enthusiasts and professionals.

Based on client's recommendation, a JavaScript Library called ReactJS was chosen as the main tool to create the interface of the project. Other tools include Bootstrap CSS library. The objective for this project was to create an interactive front-end for an application that recorded and logged flight information for pilots who either fly aircrafts with engine or, glider. This application already exists but the full form is incomplete and lacks the necessary field required by the EASA documentation [1] page 31 and 32. The client also wanted this application to be done (both the full-form and glider form) using the ReactJS framework. Both the API and the development environment for this application had been implemented prior to commissioning of this project, therefore in this thesis the main focus will be on technologies involved in creating the interfaces.

Out of the goals set for this project, creating a form that can handle the user's entry for motorized aircrafts and also display the result from database in an orderly fashion that can be further edited or deleted was accomplished. Due to various circumstances not all the objectives set for this project were accomplished. Those circumstances are further discussed in the conclusions section.

Keywords:

ReactJS, JavaScript, Bootstrap, MVC, Library, open source

TABLE OF CONTENTS

FIGURES

1 INTRODUCTION

Cavoke.com is a flight management service targeting both armature and professional pilots. This application as it pertains to this project include, flight-information logging for both glider and single/double engine plane pilots based on standards defined by the EU. Below is a list of tasks that will be accomplished during this thesis project.

- Completing the form UI done in previous project
- Integrating a entry form UI created in the previous project, with the existing application.
- Creating a POST request to the API in order to insert data from the log-form entry to database.
- The user should be able to modify and save each individual data output from where the data is being displayed.
- Creating a shorter data entry form for glider pilots. This work includes re-creating the current data-entry form with the ReactJS and API integration. The user should then be able to toggle between a glider entry-form and a full form. This option has to be also saved in user preferences.
- The result from both forms should be saved and displayed appropriately.
- Implementing flight statistics for the long-entry form (not a priority).
- The “Invoice now” and “Export log” buttons should be implemented for the long entry form.
- The pilot/co-pilot or, trainer/trainee flight hours should be saved individually/separately in designated areas.

To mimic the working application environment we are using Vagrant. In a nutshell, Vagrant is a tool for managing virtual machines (more details in section 2). Vagrant installs a virtual machine on a system which then allows the user to run/configure applications (in this case the cavokeapp) in a virtual environment. ReactJS, which is a JavaScript library, is also used to create the front-end and the Symfony PHP framework is used to create the API on the back-end.

1.1 Background

Due to advances made in JavaScript technologies that allowed for creation of various JavaScript based libraries e.g. Angular.js [2], ReactJS [3] etc, simple front-end development with static pages and simple styling are no longer the industry standard. In the past JavaScript was mainly used to create animations and interactive styling on Static pages. But, thanks to these advancements, that follow the MVC (Model-View-Controller) approach to the software development, not only developers can create interactive and modular front-end views that far exceed the traditional UIs both in terms of user experience and speed. But also utilize a more modular and compartmentalized approach to the user interface development.

Currently there are three main JavaScript libraries that mimic the MVC model in usage, Angular.js, Node.js [4] and of course ReactJS. The debate over the superiority of any of the above technologies over the other is well-established and on-going. The reason ReactJS was chosen to implement the front-end for this application was simply due to clients' requirements. ReactJS is more of a library than a framework. It provides a speedy client and server side rendering with a one-way data flow. It also allows programmers to create and utilize different components with different functionalities multiple times. Buttons, tables and fields can be created and utilized many times over. All these features make ReactJS the best candidate for this project development.

2 TOOLS

Since the author's role in this project was to create a new user interface for an already existing application and, the client provided the necessary tools and development environment, in this chapter it is mainly focused on the inner workings of ReactJS and its utilization.

2.1 Overview

ReactJS is an open-source JavaScript library which is used for creating interactive user interfaces. It was first created by Jordan Walke, a software engineer at facebook. The first utilization of ReactJS was on Facebook's newsfeed in 2011 and later on Instagram.com in 2012. One of the main advantages of ReactJS is allowing developers to create large scale web applications which can change data, without having to reload the page. This feature allows for fast, scalable and rather simple creation of large applications.

2.2 JSX

In ReactJS, it is recommended to use JSX for creating templates. JSX is an XML/HTML-like syntax used by ReactJS for creating templates. It simply allows for co-existence of XML/HTML text within JavaScript code. These syntaxes are then put through pre-processors (i.e. transpilers such as Babel) to convert/transform the HTML/XML texts found in JavaScript code into standard JavaScript objects, which can then be parsed by JavaScript engine. Using Babel's embedded editor [5] it is possible to demonstrate this feature via a simple example. Bellow we have shown a variable that contains a simple list with two items.

```
var list = (
    <ul id="list">
      <li><a href="#">Item one</a></li>
      <li><a href="#">Item two</a></li>
    </ul>
);
```

FIGURE 1. Babel transpilers demo

If the above code (Figure 1) is put through Babel, it will be transformed to JavaScript objects as demonstrated below (Figure 2). In other words, whenever JSX is included into code, it is considered as a shorthand call for `React.createElement()`.

```
var list = React.createElement(
     "ul",
     { id: "list" },
     React.createElement(
          "li",
          null,
          React.createElement(
               "a",
               { href: "#" },
               "Item one"
          )
     ),
     React.createElement(
          "li",
          null,
          React.createElement(
               "a",
               { href: "#" },
               "Item two"
          )
     )
);
```

FIGURE 2. Babel transpilers demo 2

2.3 One Way Data Flow

ReactJS utilizes a unidirectional data flow. This means that in ReactJS applications, a pyramid-like hierarchy is followed. In ReactJS applications, lower ordered child components are often nested within higher ordered parent components. The parent component will house a container that keeps track of the state of the application. In ReactJS the term “state” refers to an immutable set of variables that is stored inside the parent component. The state is owned and its scope is limited to/within the parent component where it was first declared. The state values can then be passed down to lower nested children components via read-only “props”. The children are then

able to communicate changes in state via button or form bound callbacks. In the example below this feature is demonstrated [Figure 3]

In ReactJS it is possible to create components in two different ways. The simplest way to define a ReactJS component is to write a JavaScript function. In the example provided in Figure 3, both components are defined via this method. Both “SayHi” and “App” components are functions and accept a single “props” object argument with some data and they return a ReactJS element in the JSX form. These types of Components are called “functional” because they are essentially JavaScript functions. The below snippet returns: “Hello, Sara”

```
function App() {
  return (
    <div><SayHi name="Sara" /></div>
  );
}
function SayHi (props) {
  return <h1>Hello, {props.name}</h1>;
}
ReactDOM.render(<App />, document.getElementById('root'));
```

FIGURE 3. Functional Component with props demonstration

```
class App extends React.Component {
  constructor() {
    super();
    this.state = {name: "Sara", lastName: "Smith"};
  }
  render() {
    return (
      <SayHi name={this.state.name} lastName={this.state.lastName}/>);
  }
}
function SayHi (props) {
  return <h1>Hello, {props.name} {props.lastName}</h1>
}
ReactDOM.render(<App />, document.getElementById('root'));
```

FIGURE 4. Class Component with state and props

In the example above (Figure 4) there is a parent component in “App”. App is a “Class Component” and a “State-Full Component”. Although not always necessary, Class Components can have states as well. If a Class component has state values, then it is referred to as a “state-full Component”. Inside the Constructor, state variables are initialized. The `super()` method is called if the component has a constructor. It is not possible to initialize the state before calling `super()` method because `this` before `super()` is uninitialized. In Figure 4 similar to Figure 3, the values (In this case state values) are passed down to the child Component via props. Inside the constructor, if we are going to call `this.props`, we have to pass `props` to super.

2.4 Virtual Document Object Model

DOM is an abbreviation for Document Object Model. DOMs are in essence an abstraction that take the page’s HTML structure of the page and represent it in a hierarchical/tree-like structure wrapped in an object, while maintaining the parent/child relationship between the pages.

Since DOMs were originally designed to handle static UIs, as web applications became more complex and demanding, DOMs became quite inefficient and slow because upon every change the DOM has to go through every node and look for changes and update. This style of data update detection is referred to as “Dirty Checking”. The other method is referred to as “observable” or “fast”. In the observable method, the different components of the application are responsible for listening for any accruing changes that occur. In this method the data is saved into the state and the component simply listens to events in the state for any updates.

The ReactJS engine utilizes what is called a Virtual DOM. A virtual DOM is simply a light-weight copy of the actual DOM but unlike the real DOM, VDOM does not have the ability to update or write to the page. In ReactJS a new VDOM is created every time an element is re-rendered. Right before the page is rendered, ReactJS takes a snapshot of the DOM state. This snapshot is then used against an updated VDOM before re-rendering the page again. When VDOM is updated, ReactJS uses a “diffing” algorithm to compare changes in order to detect any updates in the components and, update those elements that have changed.

2.5 NPM

NPM is a package manager for Node.js. According to the official NPM website [6] NPM is the largest software registry with approximately 3 billion downloads per week and it is installed along with Node.js. In Node.js, a “package” contains all the files required to create a “module”. The module is a JavaScript library that can be imported and used in a project therefore speeding up the development process.

NPM packages can be download via the `npm install [ package-name ]` command. The first time this command is run, NPM creates a folder named “node_modules” in the parent directory where the first installed package and all the packages to be installed in the future will be placed. When installing packages, if we add a `--save` flag is added at the end of the install command, the name and the version of the package is automatically added to the “package.json” file. The package.json file contains the name and version of all dependencies in the project.

FIGURE 5. Package.json dependencies

When it comes to sharing the project with team-mates, having a list of dependencies and their versions is very important. When this project ispushed into a repository, the “node_modules” folder can be ignored since it could contain hundreds of dependencies depending on the number of packages that have been installed. When someone clones this repository and begins working on it, all they have to do is to run the `npm install` command. All this command does is to look through package.json dependencies and install all the packages that is listed.

2.6 Webpack Module Bundler

Webpack is a very powerful tool for JavaScript developers. Webpack works by creating a dependency graph of all the dependencies in this application. The process starts from the config file (Webpack.config.js) and builds a dependency graph of all the modules that is required for the application. All the modules are then packaged into a small number of bundles to be loaded by the browser.

```
module.exports = {
  devtool: 'eval',
  entry: {
      flights: "./webApps/flightsApp.js",
      invoices: "./webApps/invoicesApp.js",
      shop: "./webApps/shopApp.js",
      reservation: "./webApps/reservationApp.js",
      flightLog: "./webApps/flightLogApp.js",
      menu: "./webApps/menuApp.js",
      admin: "./webApps/adminApp.js",
      clubs: "./webApps/clubsApp.js",
      pricingPromo: "./webApps/pricingPromoApp.js",
      account: "./webApps/accountApp.js",
  },
  output: {
    filename: '[name]Bundle.js',
    path: path.resolve(__dirname, 'www/js/genBundles')
  },
```

FIGURE 6. Webpack.config.js entry & output

Webpack setup could have one or many entry points. As mentioned before, the entry point tells Webpack where to start building the dependency graph. In this project, since the config file has already been set-up (Figure 6), all that has to be done is to set a new entry point inside `entry: {…}`. We will do so by adding `flightlog: "./webApps/flightLogApp.js",` inside the curly brackets. When `npm run build` is run, Webpack starts processing the

module at the defined entry point. From this point, it searches for other modules that depend on the entry module. This process continues until all direct/indirect dependencies have been found and configured. In a Webpack setup, also an (only one) output must be defined. After the bundles have been created, the output lets Webpack know where to place the bundle(s) and how to name them. In the example above (Figure 6) the bundle file is created via `filename: '[name]Bundle.js'` inside the output.

2.7 Symfony

Symfony is a PHP framework. Symfony aims to accelerate the creation and maintenance of a web application and also to replace recurrent coding tasks. Symfony achieves this by using a set of components and libraries

2.8 Bitbucket

Bitbucket is a web-based hosting service for projects that use either Mercurial or a Git version control. It not only allows for a safe storage of one's work but also allows for easy work flow monitoring and collaboration with other team members

2.9 Axios

Axios [9] is a Promise-based HTTP client for JavaScript which can be utilized in the front-end of this application for CRUD (create, read, update and delete) operations. Axios allows for easy creations of asynchronous HTTP requests to our REST endpoints. Axios provides the following features (according to the official website [9])

- Making XMLHTTP Requests from the browser
- Making HTTP requests from node.js
- Supporting the Promise API
- Intercepting requests and response
- Transforming request and response data
- Canceling requests
- Automatic transforms for JSON data

3 IMPLEMENTATION

3.1 Form Entry User Interface

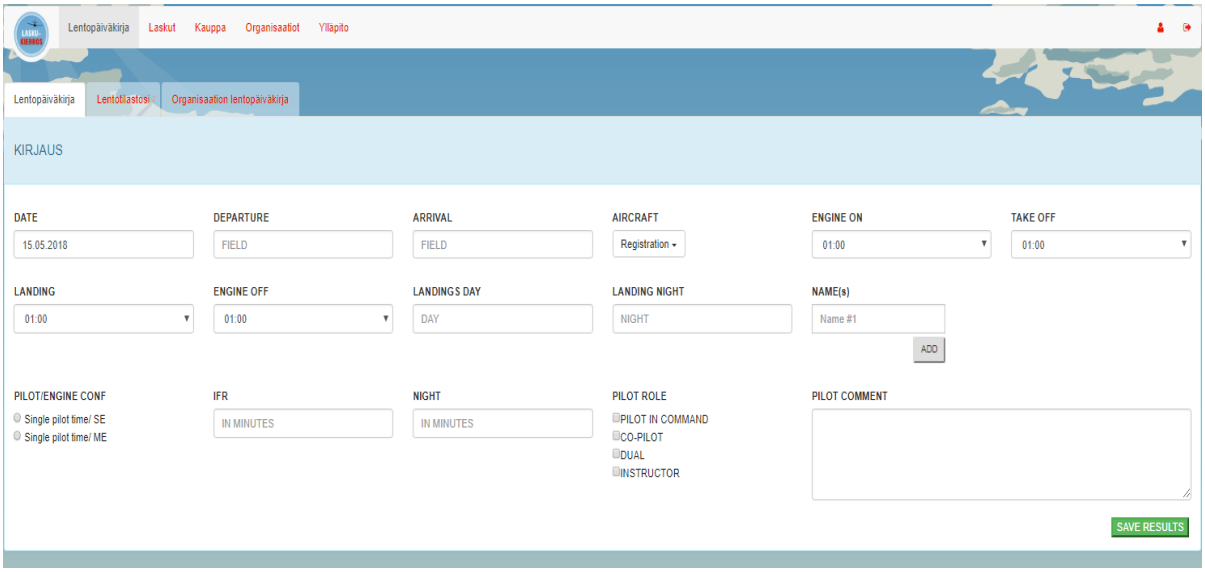

FIGURE 7. Flight Long Form User Interface

The above image shows the long-form view of the flight entry interface. This form was created based on specifications provided by the EU aviation safety agency.

3.1.1 Code-Splitting

In ReactJS applications JavaScript bundling solutions such as Webpack (chapter 2.3) or Browserify, are used to bundle all imported files and creating a single file that can be imported/included in the project. Bundles help to keep track of the applications dependencies, but as the application grows so does our bundle, especially if third-party libraries are used. And this will have an accumulating affect on the speed and performance of the application. In these types of situations it is possible to rely on a feature that bundling tools such as Webpack offer called "Code-Splitting" splitting allows to create multiple bundles that can be loaded dynamically at runtime. By using the dynamic `import()` syntax. According to ReactJS documentation [7], dynamic `import()` syntax is a ECMAScript (proposal) not currently part of the language standard. But, it is expected to be part of the language in the near future.

3.1.2 Implementation

The file structure for the flight long flight form consists of two main files. App.js witch happens to be the highest parent component of the project. Previously it was mentioned that in ReactJS applications the parent component is where the state variables are defined and later on passed down as props to children. But sometimes it is possible for children components, at least for some of them, to have constructors and state values. App.js can be divided into four parts. In the top most part using the `import ()` syntax we import all the necessary bundles are imported (Figure 8).

```
import React, { Component } from 'react';
import CSSModules from 'react-css-modules';
import PageOne from './PageOne';
import EasaFlightLog from './EasaFlightLog';
import OverallFlightStats from './OverallFlightStats';
import OrganizationFlightLog from './OrganizationFlightLog';
import moment from 'moment';
import axios from 'axios';
import styles from './App.css';
import { Tabs, Tab } from 'react-bootstrap';
import FlightLogDateField from './FlightLogDateField';
```

FIGURE 8. import syntax, App.js

App.js is a state-full component. Inside the App class a constructor is defined to initialize the state variables. Since we define a constructor for this class is defined, it is crucial that we call `super()` inside the constructor will be called before `this` is called since `this` is initialized by `super()` (Figure 9)

```
class App extends Component {
  constructor () {
    super();
    this.state = {
        date: moment().toISOString(),
        planeId: "",
        startTime:"",
        takeOffTime : "",
        landingTime: "",
        engineTimeOff: "",
        pilotComboCheckBoxes : [],
        pilotFuncTimePIC: "",
        pilotFuncTimeCoPilot: "",
        pilotFuncTimeDual: "",
        pilotFuncTimeInstructor: "",
        departureLocation: "",
        arrivalLocation: "",
        field: "",
        aircraftModel: "",
        registration: "",
        SinglePilotSingleEngine: "",
        SinglePilotMultiEngine: "",
        pilotEngineConf: "",
        names : [],
        dayLanding:"",
        nightLanding: "",
        ifrMins:"",
        nightMins:"",
        comments: "",
        pilotFlights: [],
        getRequestResults: [],
    }
```

FIGURE 9. constructor and state variables

Inside the state we define an array of variables mainly string variables, some arrays and also Date, are defined. The entire variables are empty inside the state. In the next step, inside the `render ()` function of the App component, the state and a set of setter functions are passed to the PageOne.js (Figure 10) as props. `logbook={this.state}` will allow us to access all the state values inside pageOne.js via e.g. `this.props.logBook.Date`. Similarly, it is possible to access the setting functions inside PageOne.js e.g. `this.props.handleDateChange`. `handleDateChange()` will call the `saveDate()` setter function inside App.js

```
render() {
  console.log(this.state);
  return (
      <div className="FlightLogApp">
      <Tabs defaultActiveKey={1} animation={false}>
          <Tab eventKey={1} title={ jsTranslations.tabs.flightLog }>
              <div className="panel panel-info">
                  <div className="panel-heading">
                      <h4>Kirjaus</h4>
                  </div>
                  <div className="panel-body">
                      <PageOne logBook={this.state}
                          handlePostLongFormToApi={this.handlePostLongFormToApi}
                          handleDateChange={this.saveDate}
                          handleStartTime={this.saveStartTime}
                          handleTakeOffTime={this.saveTakeOffTime}
                          ...
```

FIGURE 10. App.js render()

In ReactJS there is, a wide array of reusable, pre-made components created by third parties. These components can be downloaded via NPM and used readily inside this project. If "Date" value and how it is set as an example, the "React BootStrap based Date Picker" [8] package can simply be installed by running `npm install react-bootstrap-date-picker`. Inside PageOne.js we then import/require the component is the imported/required via `let DatePicker = require("react-bootstrap-date-picker");`

Inside the `render ()` function in PageOne.js, we simply include the date component is simply included as shown in FIGURE 11 below:

```
render() {
  return (
      <div>
          <Row>
            <form>
              <div className="form-row">
                  <div className="col-lg-2 col-md-2 col-sm-2 col-xs-12">
                    <div className="form-group">
                      <label>DATE</label>
                        <DatePicker id="flight-date-datepicker"
                          dateFormat="DD.MM.YYYY"
                          value={ this.props.logBook.date }
                          onChange={this.props.handleDateChange}
                          showClearButton={false}/>
                    </div>
                  </div>
```

FIGURE 11. DatePicker component

The Date-picker component (Figure 11) accepts a `value` and an `onChange()` parameter. Inside the `value` the state is passed and inside `onChange()` the date setter function is passed. Upon clicking on the "Date" inside the form and selecting a date (Figure 12), the saveDate() inside App.js gets called which intern sets the Date value via `setState()` (Figure 13)

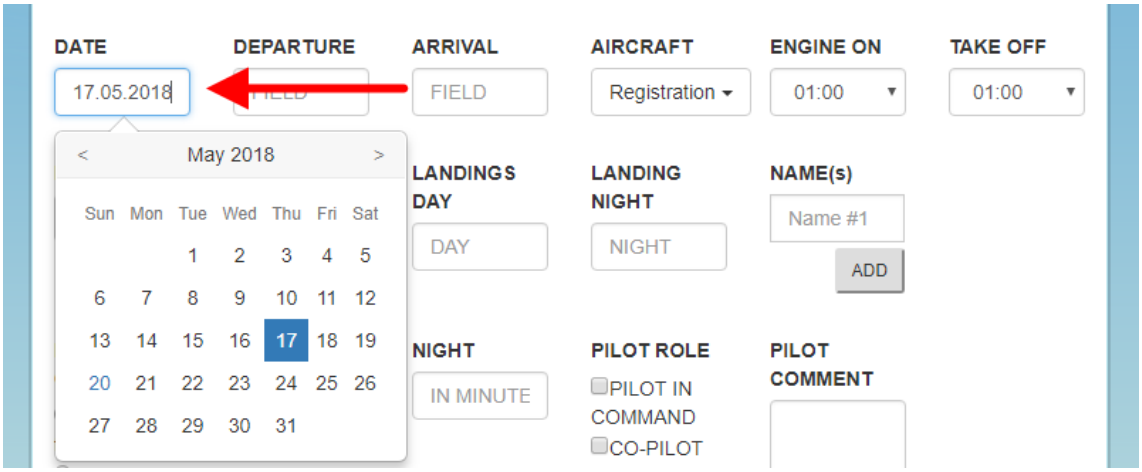

FIGURE 12 DATE select

```
saveDate(input) {
  this.setState({date: input});
}
saveDayLanding (dayLanding) {
  this.setState({ dayLanding: dayLanding.target.value})
}
```

FIGURE 13. saveDate() called

`setState()` can be thought of as a request to the ReactJS engine to update the component and its children. This action includes adding the changes to the component's state and telling ReactJS that this component needs to be updated. `setState()` in ReactJS is a primary method that is used to provoke UI changes in response to event handlers/server responses. It is important to know that ReactJS does not guarantee an immediate UI change after `setState()` is called. React may delay the UI update after `setState()` is called for a better performance hence `setState()` is considered more as a request rather that a command.

For this very reason, ReactJS does not recommend reading `this.state` right after `setState()`. Instead, a `componentDidUpdate` method or a `setState()`

callback must be used. Using these two methods will guarantee an immediate update after calling `setState()`

3.2 Post Request to the API

After parent component's state has been updated, the data must be posted to the API. Since in this project the API has already been made to handle all CRUD actions, all that has to be done is to utilize on Axios HTTP client to create and send a post request. Axios can be installed into this project via NPM and import/include the application can be imported/included inside our project. Here is how the post request looks like inside App.js

```
axios.post(newApiAddresses.flightsPost, {
  "pilotId": 1,
  "date": this.changeDateFormat(),
  "departureField": this.state.departureLocation,
  "engineOnTime": this.startTimeToGMT(),
  "takeoffTime": this.takeOffTimeToGMT(),
  "arrivalField": this.state.arrivalLocation,
  "landTime" : this.landingTimeToGMT(),
  "engineOffTime": this.engineTimeOffToGMT(),
  "planeId": this.state.planeId,
  "planeModel": this.state.aircraftModel,
  "planeReg": this.state.registration,
  "engineConfig":this.state.engineConfig,
  "pilotConfig":this.state.pilotConfig,
  "landingsDay": this.state.dayLanding,
  "landingsNight": this.state.nightLanding,
  "opConditionTimeNight": this.state.nightMins,
  "opConditionTimeIFR": this.state.ifrMins,
  "pilotFuncTimePIC": this.state.pilotFuncTimePIC,
  "pilotFuncTimeCoPilot": this.state.pilotFuncTimeCoPilot,
  "pilotFuncTimeDual": this.state.pilotFuncTimeDual,
  "pilotFuncTimeInstructor": this.state.pilotFuncTimeInstructor,
  "pilotComments": this.state.comment,
  "flightType" : "No React implementation yet",
  "flightTypeId": null
 })
.then(function (response) {
  console.log(response);
})
.catch(function (error) {
  console.log(error);
});
```

FIGURE 14. Axios POST request

Axios.post accepts two parameters; the first parameter is the API URL for POST that has been defined in file flightLogReact.html.php as shown in FIGURE 15 below:

```
var newApiAddresses = {
    flightsGet: "<?php echo $view['router']->url('flybook_test_api_v1_apiflights_getflights'); ?>",
    flightsPost: "<?php echo $view['router']->url('flybook_test_api_v1_apiflights_post'); ?>",
    flightsPut: "<?php echo $view['router']->url('flybook_test_api_v1_apiflights_put'); ?>"
}
```

FIGURE 15. API addresses for CRUD operations

The second parameter inside the request is an Array of variables to be sent and set to the database. Since all these variables are state variables, the values are fetched via `this.state.[name]`. After these two parameters have been set, the `then()` method returns a `Promise`. The `then()` method takes up to two arguments: callback functions to indicate the success or failure of the `Promise`

3.3 Fetch and Display

3.3.1 GET request

In order to be able to display the user-entry first, a get request must be created to the API. Similar to the POST request, using the Axios HTTP client a new get request is created.

```
handleGetRequestFromApi () {
  axios.get(newApiAddresses.flightsGet).then(response => {
    const pilotFlights = response.data.map(f => {
      f.date = moment.utc(f.date, "DD.MM.YYYY").clone();
      return f;
    });
    this.setState({ pilotFlights: pilotFlights });
  });
}
componentDidMount()
{
  this.handleGetRequestFromApi();
}
```

FIGURE 16. Get Request

In the above request a GET request is sent to the pre-defined API address at `newApiAddresses.flightsGet`. Using a map function the output is looped through

and the outputs are stored as an array. Inside the map function we also do any conversions necessary for Date and Time values are made. After the data is fetched, we set the state value for pilot flights with our newly created array are set. Calling the Get request inside `componentDidMount()` will ensure that the get method is invoked immediately after the component is mounted. When we call `setState()` is called inside from `componentDidMount()`, this will cause an extra rendering. This rendering will take place before the browser updates the screen. Although this method is not recommended, in some cases it can be necessary especially for cases such as modals and tooltips.

3.3.2 Display & Edit View

```
<EasaFlightLog flights={this.state.pilotFlights}
  handleEditFlight={ this.handleEditFlight }
  deleteFlight = {this.deleteFlight}
  organizationData = { organizationData }
/>
```

FIGURE 17. EasaFlightLog

In order to display the user data a full advantage of modularity of ReactJS must be taken. `<EasaFlightLog />` is the child of the parent component `<App />`. Inside `<EasaFlightLog />` exist `<EasaLogRow />` which contains a set of individual cells, each containing a component representing either the view or edit version of a single data point.

```
<td>
  <FlightLogDateField
    date={this.state.flightData.date}
    editModeActive={this.state.editModeActive}
    onChange={this.editDate.bind(this)}
  />
</td>
<td>
  <FlightLogTakeOffField
    takeOffField = {this.state.flightData.departureField}
    editModeActive={this.state.editModeActive}
    onChange={this.editTakeOffField.bind(this)}
  />
</td>
<td>
  <EngineOnTimeField
    engineOnTime = {this.state.flightData.engineOnTime}
    editModeActive={this.state.editModeActive}
    onChange={this.editEngineOnTimeField.bind(this)}
  />
</td>
```

FIGURE 18. EasaLogRow.js

Inside `<EasaFlightLog />` (Figure 17) an array is passed containing flights fetched from database, two functions, one for editing the dataset and another to delete. `handleEditFlight={this.handleEditFlight}` (Figure 19) will receive a newly edited flight log from `<EasaFlightRow />` passed up as a parameter and use the received set of data to reset the pilot Flight array with the newly set state. `deleteFlight = {this.props.deleteFlight}` will simply pass up the flight id to `App.js` and passes a request to the delete function in the API

```
handleEditFlight (newFlightData)
{
  let id = 3;
  let pilotFlights = this.state.pilotFlights.map(pf => {
    if(pf.id === newFlightData.id)
    {
      return newFlightData; // set the new flightData into the pilotFlight arr
    }
    else
    {
      return pf;
    }
  }); // clone the array and objects

  this.setState({ pilotFlights: pilotFlights });
  this.handlePutRequestToApi(id);
}
```

FIGURE 19. handleEditFlight()

As mentioned earlier, each table cell representing a data point is represented as a separate state –fewer components (Figure 18) that will accept a state variable `editModeActive={this.state.editModeActive}` as props that will determine whether the cell component (Figure 20) will display the view mode or edit mode.

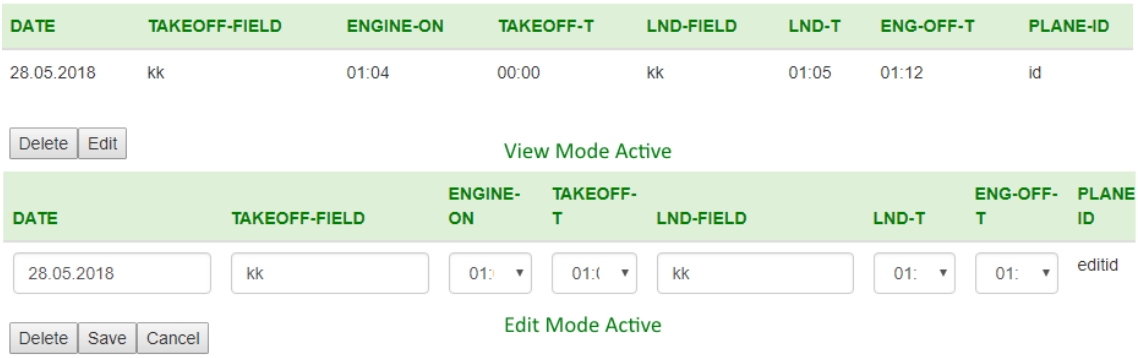

FIGURE 20. Table View/Edit Mode

When the user presses the edit button, a function toggleEditMode () is called, which in turn sets the editModeActive state variable to either true or false. The editModeActive is passed as props to each individual table cell component. If for instance one needs to consider how the Date view mode is set inside the date component (Figure 21) the following shows how the component toggles between the edit and view mode

```
class FlightLogDateField extends Component {
  constructor (props) {
    super(props);
  }
  render () {
    let output;
    if(this.props.editModeActive)
    {
      output = (
        <div>
          <DatePicker id="flight-date-datepicker" onChange={this.props.onChange}
                dateFormat="DD.MM.YYYY" value={ this.props.date.toISOString() }
                showClearButton={false} />
        </div>
      );
    }
    else
    {
      output = (<div>{this.props.date.format('DD.MM.YYYY')}</div>);
    }
    return output;
  }
}
```

FIGURE 21. Date component View/edit mode toggle

4 CONCLUSION

The main objectives to be accomplished for this thesis have been listed in the Introduction chapter. Out of these tasks, I managed to accomplish four of them and the last three tasks (the other being optional) where not accomplished. The not accomplished tasks are listed below.

- Creating a shorter data entry form for glider pilots. The user should then be able to toggle between a glider entry-form to a full form. This option also has to be saved in user preferences (Note: this section is the exact replica of the long-form section but shorter)
- Implementing flight statistics for the long-entry form (not a priority)
- The “Invoice now” and “Export log” buttons should be implemented for the long entry form
- The pilot/co-pilot or, trainer/trainee flight hours should be saved individually/separately into designated areas

The main reasons behind the project being incomplete are first, my lack of knowledge about the technologies utilized in this project as well as the lack of adequate planning and time management. Another reason that can be listed as a catalyst would be continual changes that needed to be made, parts and features being added and later removed. Since the beginning, there was no agreement on how the UI should look like and what it should include. Another factor that added to the problem was me continuing to work on the development version of ReactJS as well as creating the API for the POST and GET request instead of installing and working with the development environment created by the customer and included API.

Doing this project I gained valuable knowledge about developing with a very popular JavaScript library in ReactJS as well as the workings of virtual environments such as Vagrant and how a JavaScript library such as react can be utilized in such an environment. I also learned a lot about the version control software Bitbucket and how such an environment can be utilized for effective team work.

5 REFERENCES

1. EASA (European Aviation Safety Agency). Date of retrieval 08.05.2018, https://www.easa.europa.eu/sites/default/files/dfu/AMC%20+%20GM%20to%20Regulation%201178%20of%202011-Revision-June%202016_V03.pdf
2. AngularJS official website. Date of retrieval 08.05.2018, https://angularjs.org/
3. ReactJS Official Website. Date of retrieval 08.05.2018, https://reactjs.org/
4. NodeJS Official Website. Date of retrieval 08.05.2018, https://nodejs.org/en/
5. Babel Official Documentation. Babel transpilers. Date of retrieval 16.05.2018, https://babeljs.io/repl/
6. NPM Official Website. Date of retrieval 18.05.2018, https://www.npmjs.com
7. ReactJS Official Documentation. Date of retrieval 20.05.2018, https://reactjs.org/docs/code-splitting.html
8. React Bootstrap Date picker. Date of retrieval 20.05.2018, https://www.npmjs.com/package/react-bootstrap-date-picker
9. Axios HTTP Request. Date of retrieval 25.05.2018, https://www.npmjs.com/package/axios