МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Объектно-ориентированное программирование»

на тему «Расписание движения транспорта(аэропорта)**»**

Выполнил:

студент группы 10702216 Анисович М.А.

Принял: ст.пр. Иванченко В.В.

Минск 2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

по дисциплине «Объектно-ориентированное программирование»

на тему «Расписание движения транспорта(аэропорта)**»**

Выполнил:

студент группы 10702216 Анисович М.А.

Принял: ст.пр. Иванченко В.В.

Минск 2019

**Содержание**

[**Введение** 4](#_Toc8995908)

[**1. Техническое задание на предмет разработки** 5](#_Toc8995909)

[**2. Обоснование используемых языков и технологий.** 6](#_Toc8995910)

[**3. Моделирование и проектирование программного обеспечения** 8](#_Toc8995911)

[**4. Реализация программного обеспечения** 10](#_Toc8995912)

[**5. Развёртывание и тестирование программного обеспечения** 21](#_Toc8995913)

[**Заключение** 23](#_Toc8995914)

[**Список литературы** 24](#_Toc8995915)

[Приложение 25](#_Toc8995916)

# **Введение**

Сегодняшний мир невозможно представить без перелётов. Авиатранспорт используют как средство передвижения как для работы, так и для туризма и путешествий. В нашей интенсивной жизни огромное значение имеет планирование своего времени. Поэтому выбор удобного времени для путешествия важная задача и, конечно же, средства для её выполнения должны быть максимально доступны.

Для пользования пассажирами расписание может быть вывешено или нарисовано на табличке у места ожидания транспорта, в аэропортах информация о рейсах обновляется и отображаться на табло.

Для максимальной доступности и удобства расписание можно разместить на веб-сайте. Приложение для расписания способно предоставить всю необходимую информацию по интересующему маршруту: аэропорта вылета и прилёта, время отправления самолётов и их модель.

Сайт-расписание – его главный источник информации для планирования своего маршрута.

Меня заинтересовала эта тема, поэтому я решил создать такое приложение, которое предоставило бы пользователю возможности, перечисленные выше.

# **Техническое задание на предмет разработки**

Цели, которые необходимо достигнуть в процессе разработки сайта-расписания: разработать веб-приложение, с понятным, простым и удобным интерфейсом, которое позволит пользователю выбирать города его маршрута с удобным вводом на любом языке, необходимую дату, учитывающее специфику трансферных перелётов и, самое главное, предоставляющее актуальную информацию.

Разрабатывая функционал, я выделил для себя следующие необходимые детали, которые необходимо реализовать в моём веб-приложении:

* **Города маршрута.** Это один из главных компонентов любого расписания. Важно реализовать простоту ввода для пользователя, предоставляю возможность вводить информацию на любом языке.
* **Дата**. Для облегчения ввода даты вылета необходимо предоставить пользователю удобный выбор даты в виде календаря, который учитывает актуальную дату и позволяет выбирать дни на несколько месяцев вперёд.
* **Трансферные перелёты.** Не всегда возможно попасть напрямую из одного города в другой. Зачастую необходимо совершать пересадку в промежуточном аэропорту. Если будущий пассажир не знает точно, имеются ли прямые рейсы, необходимо ему подсказать о возможности выбора трансферных рейсов.
* **Непосредственно расписание.** Предоставление актуального расписания – основная задача данного приложения. Оно представляет собой таблицу со всей необходимой информацией.
* **Возможность быстро менять местами города маршрута для планирования обратного пути.**

# **2. Обоснование используемых языков и технологий.**

Основным языком для создания проекта был выбран JavaScript. из преимуществ данных языка можно выделить:

* Быстрый для конечного пользователя: написан для клиентской стороны, для поддержки веб-сервера не требуется поддержка. Он также не нуждается в компиляции на стороне клиента, что дает ему определенные преимущества скорости.
* Простота: JavaScript относительно прост в освоении и реализации. Он использует модель DOM, которая обеспечивает множество предустановленных функций для различных объектов на страницах, что делает его легким для разработки сценария для решения пользовательской цели.
* Универсальность: JavaScript отлично работает с другими языками и может использоваться в самых разных приложениях.

React это библиотека для создания пользовательских интерфейсов. Одной из ее отличительных особенностей является возможность использовать **JSX**, язык программирования с близким к HTML синтаксисом, который компилируется в JavaScript.

При составлении React компонентов используется HTML. **HTML** — язык разметки гипертекста, который сделал интернет таким, каким мы его знаем и любим. Благодаря этому инструменту сайты выглядят красиво и современно, а также обеспечивается удобство их использования. Язык HTML просто компонует элементы веб-страницы в удобный для пользователя вариант. HTML считается самым демократичным языком, так как он оптимальным образом поддерживается и интерпретируется Интернет-проводниками.

Для описания внешнего вида документа используется **CSS** и **Bootstrap**. Как таковым отдельным языком программирования CSS нельзя считать, так как он не имеет значения без документов, построенных на HTML или XHTML. Каскадные таблицы стилей описывают правила форматирования элементов с помощью свойств и допустимых значений этих свойств. Для каждого элемента можно использовать ограниченный набор свойств, остальные свойства не будут оказывать на него никакого влияния. Bootstrap это фреймворк из трёх языков HTML/CSS/JS. Благодаря большому функционалу верстать сайты становится легко и быстро, ну конечно если во всем разобраться. У данного фреймворка можно выделить следующие плюсы:

1. *Быстрота верстки* — большое количество готовых компонентов даёт возможность не останавливаться на обыденностях.
2. *Адаптивность* — возможность настраивать размеры блоков сайта в зависимости от ширины устройства, как для компьютера так и для телефона.
3. *Популярность* — из-за которой существует большое количество статей и уроков, а также форумов. Поэтому по любому пустяку, в котором вы сомневаетесь можете найти ответ на просторах интернета или задать вопрос на форуме.
4. Bootstrap можно использовать для создания сайтов с *различными CMS* — WordPress, Joomla, Opencart.

База данных для заполнения расписания получается путём отправления **API** запросов и формирования json файла с актуальными данными. API (Application Programming Interface) — это интерфейс программирования, интерфейс создания приложений. Если говорить более понятным языком, то API — это готовый код или структура данных для упрощения жизни программисту.

**API Яндекс.Расписаний** — инструмент программного доступа к данным сервиса о маршрутах и времени движения транспортных средств.

API организован в соответствии с принципами REST. Взаимодействие с API осуществляется по протоколу HTTPS посредством GET-запросов. Сервис возвращает данные в форматах: JSON, XML.

# **3. Моделирование и проектирование программного обеспечения**

Веб-приложение — [клиент-серверное](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80) приложение, в котором [клиент](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) взаимодействует с сервером при помощи [браузера](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80), а за сервер отвечает — [веб-сервер](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80). За клиентсукую часть отвечает React, он в асинхронном режиме отправляет API запросы для получения актуальной базы данных.

Клиентская часть React приложения

API база данных

Рис. 1 Архитектура веб-приложения.

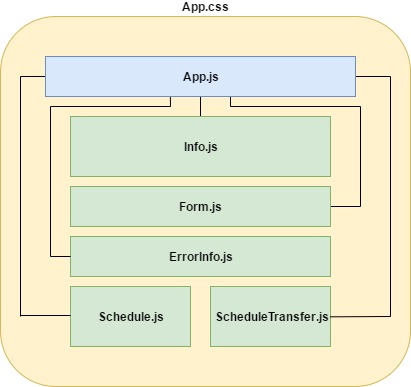


Рис. 2 Архитектура React приложения.

На схеме представлены следующие страницы:

* App.js – основной компонент React приложения, который компонует в себе все остальные созданные компоненты, принимает введённые пользователем данные, обрабатывает их и отправляет на вывод в другие компоненты. Создаётся автоматически при создании React проекта.
* Info.js – компонент, который выводит текстовую информацию.
* Form.js – компонент, предоставляющий пользователю форму для ввода. В себе содержит: поле для ввода города отправления, города прибытия, форма для ввода даты с возможностью выбрать её в календаре, кнопка, меняющая местами города отправки и прибытия, checkbox для трансферных рейсов, кнопка, отправляющая введённые данные на обработку для дальнейшего вывода в таблицу.
* Schedule.js – компонент, принимающий информацию о прямых рейсах из основного компонента и выводящий её в виде таблицы.
* ScheduleTransfer.js – компонент, принимающий информацию о трансферных рейсах из основного компонента и выводящий её в виде таблицы.
* Error.js – компонент, который отображает необходимую информацию в случае, если пользователь ввёл некорректные данные.
* App.css – таблица стилей для оформления внешнего вида приложения.

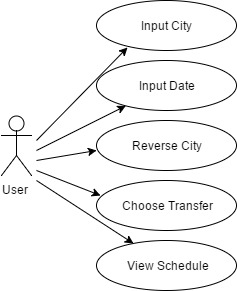


Рис. 3 Use-case диаграмма.

На use-case диаграмме отображают уровни доступа в приложении, которыми обладают пользователи.

Т.к. приложение-расписание должно быть максимально доступно всем людям и для всех предоставлять одинаковые возможности, был реализован один, общий для всех, уровень доступа, который предоставляет возможность вводить необходимые города, дату, тип рейса (прямой или трансферный), менять местами города, для удобного планирования обратного пути, и возможность видеть само расписание.

# **4. Реализация программного обеспечения**

Точкой входа для React приложения является файл public/index.html он предстовляет из себя простой html-документ, который имеет следующий блок:

<div id="root"></div>

В этом теге работает всё React приложение. Далее идем в файл src/index.js - наш главный файл реакта приложения, здесь мы подключаем сам react, react-dom (для работы с web-приложениями), подключаем стартовый компонент App и указываем в каком теге из файла public/index.html будет работать наше приложение.

ReactDOM.render(<App />, document.getElementById('root'));

Запустив приложение, мы попадаем на стартовую страницу:

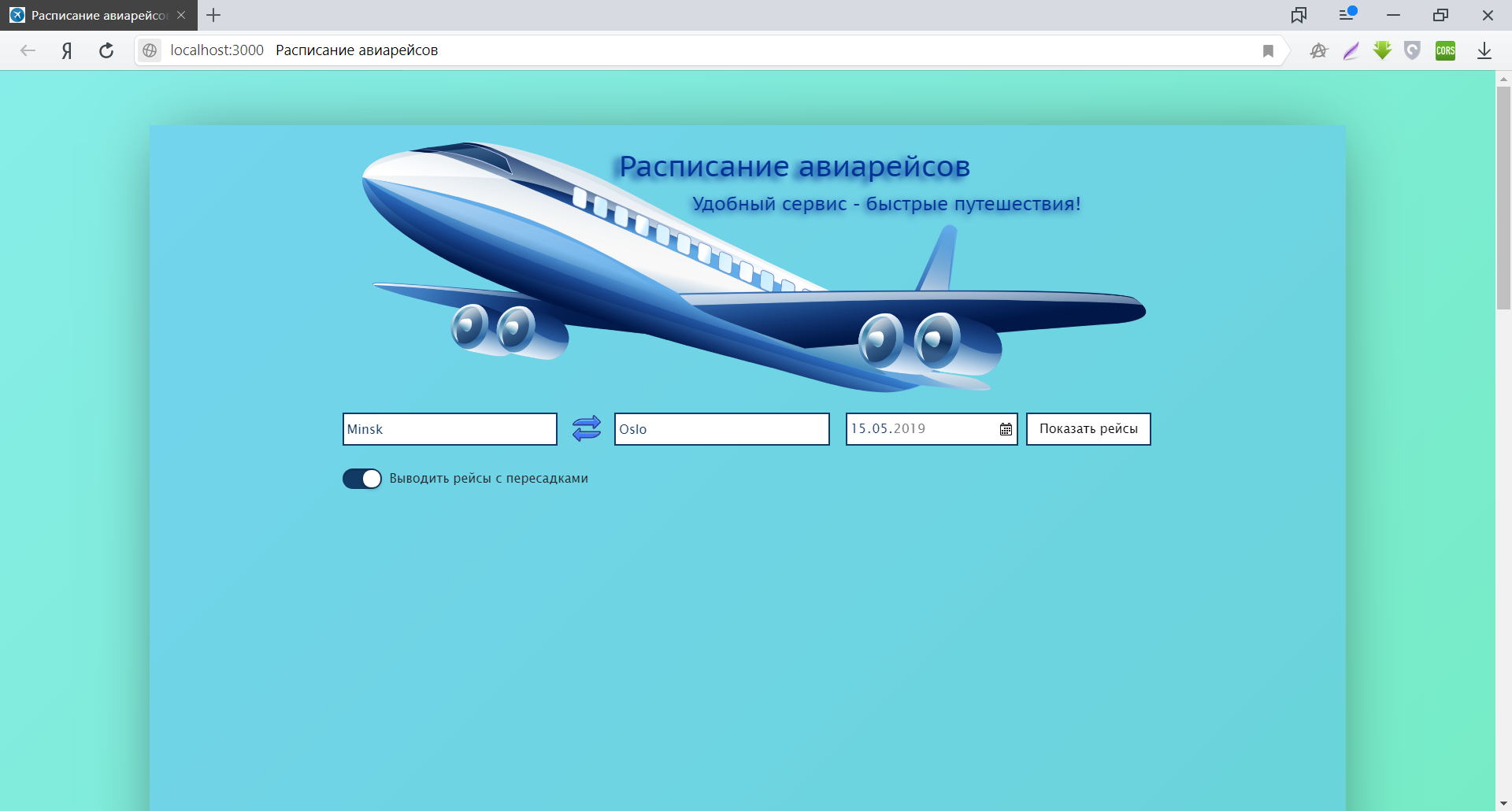


Рис. 4 Стартовая страница.

На стартовой странице отображаются первые компоненты приложения. Это компоненты Info и Form, помещённые внутрь компонента App. Также, для создания дизайна, эти компоненты помещены в иерархию тегов <div>. Для стилизации блоков были использованы Css стили и Bootstrap.

В компоненте Form находится тег <form> устанавливает форму на веб-странице. Форма предназначена для предоставления удобного интерфейса ввода пользовательских данных. Данная форма состоит из тегов <input> различного типа: button, checkbox, date и text, а также кнопки button.

По умолчанию в текстовых формах установлено значение городов. Эти значения формы получают из переменных класса Form.

this.state = {

fromCity: 'Minsk',

toCity: 'Oslo',

...

};

Сам input имеет следующий вид:

<input type="text" name="fromCity" value={fromCity} onChange={this.handleChange} placeholder="Город отправления"/>



Рис. 5 Значение input .

При вводе пользователем текста вызывается метод handleChange, он даёт возможность изменять значение value.

handleChange = e => {

e.preventDefault();

this.setState({

[e.target.name]: e.target.value

})

}



Рис. 6 Введённые пользователем города.

Пользователю предоставлена возможность поменять местами город отправления и прибытия. Для этого был использован input типа button.

<input type="button" onClick={this.reverseCity}/>

Вызовом метода reverseCity города меняются местами.

reveresCity(){

const nowFromCity = this.state.fromCity;

const nowToCity = this.state.toCity;

this.setState({

fromCity: nowToCity,

toCity: nowFromCity

});

}

Чтобы пользователю было интуитивно понятно предназначение кнопки, её стандартный вид был заменён на картинку стрелок. Стиль кнопки был изменён с помощью CSS.

input[type="button"] {

border: 1px solid rgba(0, 0, 0, 0);

outline: none;

...

background: url(https://www.shareicon.net/data/2016/05/24/770231\_arrows\_512x512.png) no-repeat 3px 1px;

background-size: auto 90%;

}

Пользователь может выбрать дату с помощью input типа date. Это значительно удобнее ввода даты вручную.

<input type="date" name="allDate" value={allDate} onChange={this.handleChange} max={maxDate} min={minDate}/>

В поле ввода добавлена иконка календаря, при нажатии на которую появляется выпадающий календарь.



Рис. 7 Поле для ввода даты.

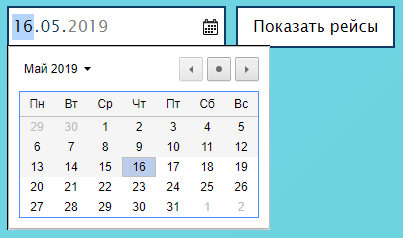


Рис. 8 Выпадающий календарь.

В календаре для выбора даты доступен промежуток от текущего дня, который получаем по текущей дате компьютера, и на 3 месяца вперёд. Значения даты хранятся в строковом виде yyyy-mm-dd как переменная класса Form.

constructor(props) {

super(props);

this.state = {

...

allDate: '',

minDate: '',

maxDate: '',

...

};

В строковый вид дату переводим с помощью метода componentWillMount, который устанавливает значение текущей даты и границы дат.

componentWillMount() {

const date = new Date();

const day = date.getDate();

const month = (date.getMonth() > 9) ? date.getMonth()+1 : "0" + (date.getMonth()+1);

const year = date.getFullYear();

const dateMax = new Date(year, +month + 2, day)

const maxDay = dateMax.getDate();

const maxMonth = (dateMax.getMonth() > 9) ? dateMax.getMonth()+1 : "0" + (dateMax.getMonth()+1);

const maxYear = dateMax.getFullYear();

this.setState({

allDate: `${year}-${month}-${day}`,

minDate: `${year}-${month}-${day}`,

maxDate: `${maxYear}-${maxMonth}-${maxDay}`

})

}

Элемент checkbox предназначен для выбора отображения прямых или трансферных рейсов. Тег <lebel> выводит текст возле checkbox.

<label className="checkbox">

<input type="checkbox" defaultChecked={isCheck} onChange={this.handleChangeCheckbox} name="transfer"/>

<div className="checkbox\_\_text">

Выводить рейсы с пересадками

</div>

</label>

По умолчанию значение установлено true и является переменной класса Form.

constructor(props) {

super(props);

this.state = {

...

isCheck: true

};

Так же реализован метод handleChangeCheckbox, который меняет значение булевской переменной на противоположное.

handleChangeCheckbox = e => {

this.setState({

isCheck: !this.state.isCheck,

});

}

Нажатие фиксируется как при клике по самому checkbox, так и по тексту.



Рис. 9 CheckBox в состоянии true.



Рис. 10 CheckBox в состоянии false.

Кнопка button, созданная в теге <button>, при нажатии вызывает метод класса App, переданный компоненту Form. После нажатия происходит обработка введённых пользователем данных.

Кнопка имеет разные стили в обычном состоянии и при наведённом курсоре. Свойство hover CSS стилей описывает вид кнопки при наведённом курсоре.

button {

background-color: white;

outline: none;

color: black;

border: 2px solid #113a65;

padding: 7px 15px;

}

button:hover {

background-color: #113a65;

color: white;

}



Рис. 11 Обычное состояние кнопки.



Рис. 12 Кнопка при наведении курсора.

После нажатия кнопки начинается обработка данных. Метод получает данные и записывает их в переменные для дальнейшей работы. Для получения данных расписания будет отправлен запрос с помощью индивидуального API ключа Яндекс.Расписание. Запрос отправляется асинхронно, чтобы не перезагружать страницу браузера. Это реализуется использованием конструкции async/await. Await запрос мы отправляем с помощью URL ссылки. Запрос имеет следующий синтаксис:

https://api.rasp.yandex.net/v3.0/search/ ?

from=<код станции отправления>

& to=<код станции прибытия>

& [lang=<язык>]

& [apikey=<ключ>]

& [date=<дата>] // выбор пользователем

& [transport\_types=<тип транспорта>] // plane

& [system=<система кодирования параметров to и from>] // iata

& [transfers=<признак запроса маршрутов с пересадками>] // зависит от выбора

Пользователь может вводить города на любом языке. Запрос на Яндекс.Переводчик принимает текст и переводит его на английский язык. Если город отправления введён на русском языке, то land параметр Яндекс.Расписания принимает русский язык.

В запросе используется система кодирования IATA. Получение кода города происходит после отправление запроса на https://iatacodes.org/api/v6/cities?api\_key=<ключ>. В соответствии с введённым городом мы получаем его код, который передаём во from и to параметр Яндекс.Расписания.

Когда получены все параметры для запроса, он возвращает информацию в формате json.

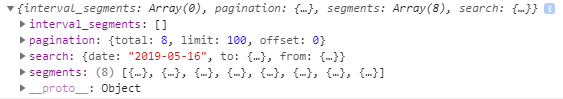


Рис. 13 Ответ на запрос в json формате.

Если параметр transfers запроса имеет значение false, а рейсы доступны только с пересадками, то значение pagination.total будет равно нулю. В случае, если имеются прямые рейсы, то булевское значение segments[0].has\_transfers будет false, а параметр запроса transfers необходимо установить true.

Метод getSchedule после корректной работы всех запросов устанавливает значение переменных класса App.

constructor(props) {

super(props);

this.state = {

segmentsInfo: false, // значение по умолчанию

segmentsInfoTransfer: false, // значение по уполчанию

...

}

}

Устанавливаемые значения – sigments. Sigments является массивом данных, находящихся в json.

Отображение таблиц расписания происходит в компонентах Schedule (в случае прямых доступности прямых рейсов) и ScheduleTransfer(в случае трансфенрых перелётов).

В компоненте Schedule метод renderTable выполняет функцию отрисовки таблицы, пробегая по sigmentsInfo и добавляя в таблицу строку с информаций из каждого элемента массива.

renderTable =() => {

let row;

if(this.props.segmentsInfo){

row = this.props.segmentsInfo.map((item, index) => (

<tr key={index}>

<td>{item.thread.number}</td>

<td>{item.thread.vehicle}</td>

<td>{item.from.title} - {item.to.title}</td>

<td>{item.departure.substr(11, 8) + ' ' + item.departure.substr(19, 6)}</td>

<td>{item.arrival.substr(11, 8) + ' ' + item.arrival.substr(19, 6)}</td>

</tr>

))

return(<table>

<tbody>

<tr>

<th>Номер рейса</th>

<th>Самолёт</th>

<th>Маршрут</th>

<th>Время вылета</th>

<th>Время прибытия</th>

</tr>

{row}

</tbody>

</table>)

}

}



Рис. 14 Отображение таблицы прямых рейсов.

В компоненте ScheduleTransfer метод renderTable выполняет функцию отрисовки таблицы путём добавления информации как столбец таблицы, чтобы вывести вместе рейсы, описывающие трансфер между точками маршрута.

renderTable =() => {

let number, planeName, route, departureTime, arrivalTime;

if(this.props.segmentsInfoTransfer){

number = this.props.segmentsInfoTransfer.map((item, index) => (

<tr key={index}>

<td>

<tr>With transfer</tr>

<tr>{item.details[0].thread.number}</tr>

<tr>{item.details[2].thread.number}</tr>

</td>

</tr>

))

planeName = this.props.segmentsInfoTransfer.map((item, index) => (

...

route = this.props.segmentsInfoTransfer.map((item, index) => (

...

departureTime = this.props.segmentsInfoTransfer.map((item, index) => (

...

arrivalTime = this.props.segmentsInfoTransfer.map((item, index) => (

...

return(<div className="table-responsive"><table className="table">

<tbody>

<tr>

<th>Номер рейса</th>

<th>Самолёт</th>

<th>Маршрут</th>

<th>Время вылета</th>

<th>Время прибытия</th>

</tr>

<td>{number}</td>

<td>{planeName}</td>

<td>{route}</td>

<td>{departureTime}</td>

<td>{arrivalTime}</td>

</tbody>

</table>

</div>)

}

}



Рис. 15 Отображение таблицы трансферных рейсов.

Компонент ErrorInfo выводит ошибки, которые могут возникнуть при работе сайта. Переменные, в которые в строковом виде принимают информацию о ошибках, являются переменными класса App, которые передаются в компонент ErrorInfo.

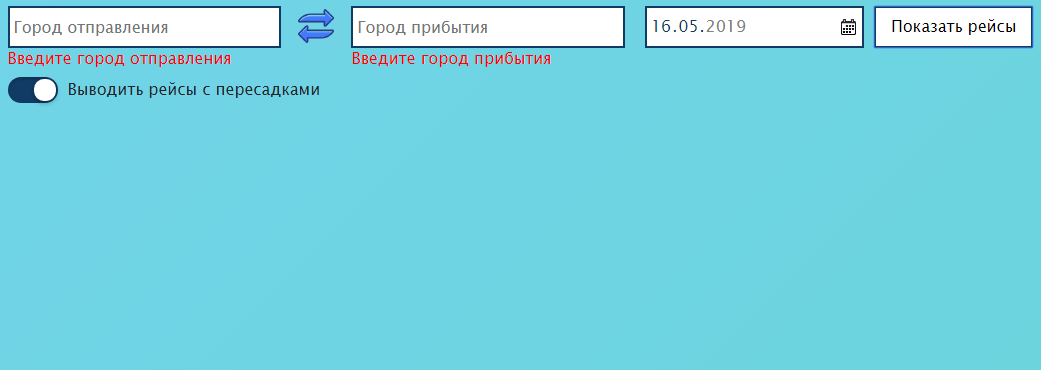


Рис. 16 Ошибка при пустых полях ввода.



Рис 17. Ошибка при вводе несуществующего города.

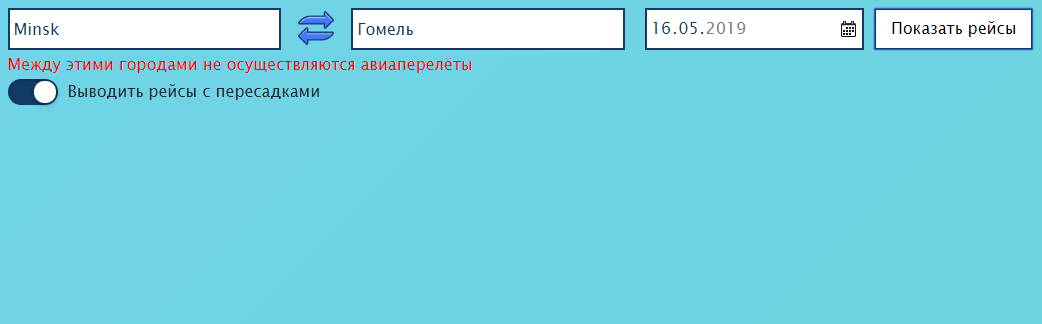


Рис. 18 Ошибка при несуществующем маршруте.

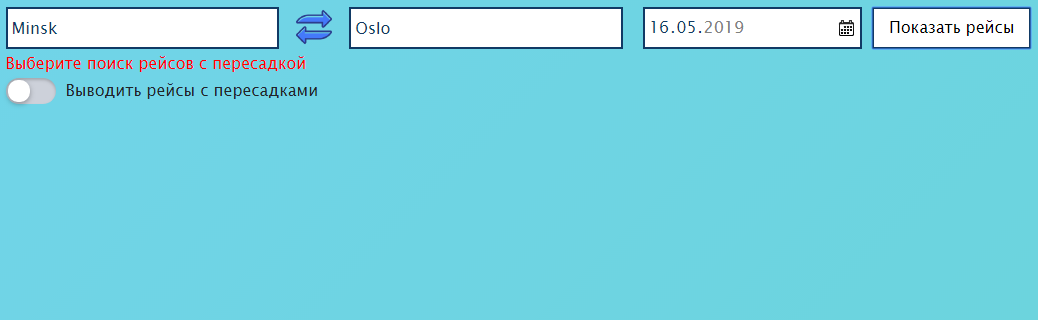


Рис. 19 Ошибка при отсутствии прямых рейсов.

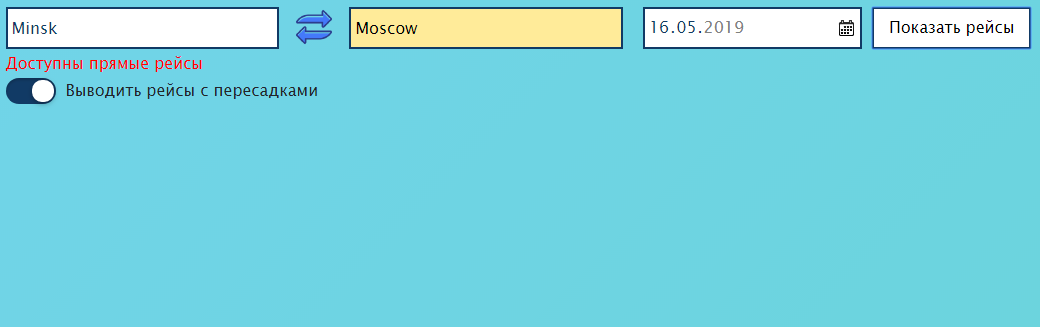


Рис. 20 Ошибка при доступности прямых рейсов.

# **5. Развёртывание и тестирование программного обеспечения**

Для запуска и тестирования приложения во время разработки был установлен локальный сервер с помощью консольной команды npm start. Этот сервер компилировал JavaScript код и запускал его на локальном хосте http://localhost:3000/.

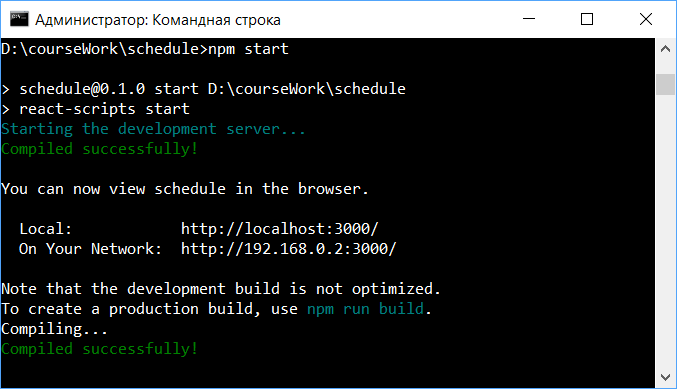


Рис. 21 Запуск и работа локального сервера.

После окончательной отладки приложение было развёрнуто и готово для распрастронения и запуска на любом компьютере. С помощтю команды npm build оно было собрано и помещено в папку build. Для запуска сайта достаточно запустить index.html.

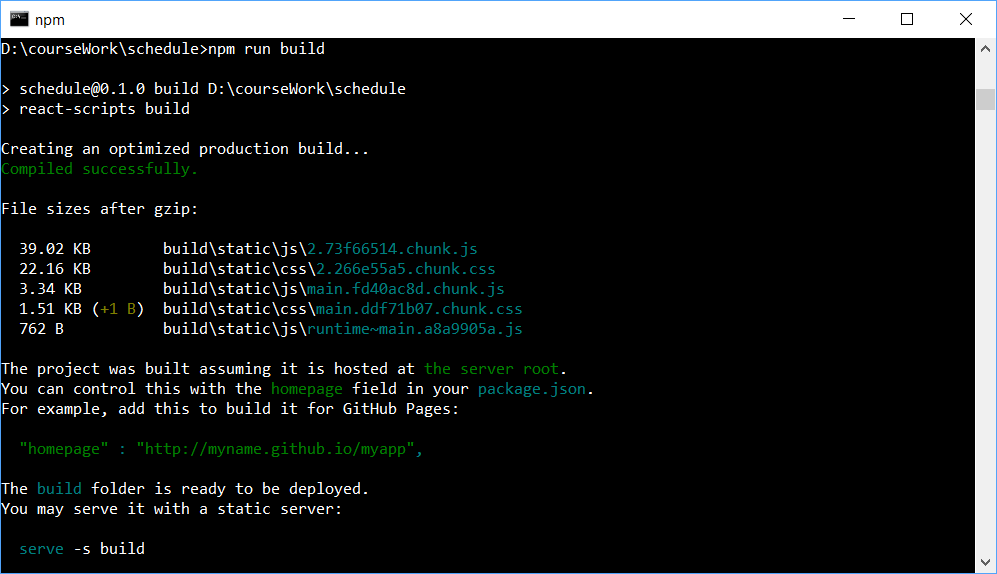


Рис. 22 Сборка готового приложения.

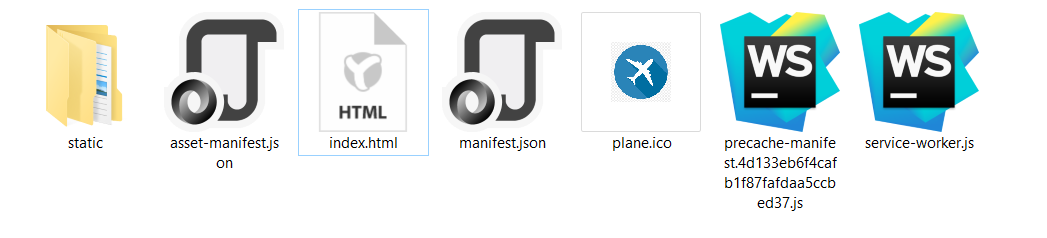


Рис. 30 Файлы развёрнутого приложения.

Для корректной работы запросов в приложении, не размещённом на хостинге с доменом типа http://www.mydomain.com/, необходимо установить в браузере расширение для CORS, позволяющее отправлять запросы.

# **Заключение**

Поставленные цели, я считаю, были достигнуты. Разработано приложение с простым, понятным и удобным интерфейсом, имеющиее весь необходимый функцианал для сайта-расписания. Хороший дизайн, рабочая система, актуальная информация и другие аспекты влияют на то, будут ли пользователи обращаться именно к этому сайту.

В результате работы были закреплены и улучшены значния о HTML разметке, таблице стилей CSS. Освоена библиотека React языка программирования JavaScript. Изучены принципы работы с асинхронными забросами, получение данных по API ключу.

# **Список литературы**

1. «JavaScript. Подробное руководство» - Флэнаган Дэвид
2. «Объектно-ориентированное конструирование программных систем» - Бертран Мейер.
3. https://reactjs.org - официальный сайт React
4. http://htmlbook.ru/ - Справочник по html и css.

# Приложение

Листинг программы

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import \* as serviceWorker from './serviceWorker';

import "bootstrap/dist/css/bootstrap.min.css";

import "./App.css"

ReactDOM.render(<App />, document.getElementById('root'));

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

**App.js**

import React from "react";

import Info from "./components/Info";

import Form from "./components/Form";

import Schedule from "./components/Schedule";

import ErrorInfo from "./components/ErrorInfo";

import ScheduleTransfer from "./components/ScheduleTransfer";

import "./App.css"

const API\_KEY\_SCHEDULE = "06d78f5c-1ffe-4f3d-b3a8-7e69b952d5d4";

const API\_KEY\_CITY\_CODE = "d5574245-068f-411e-bd9b-48b9715d1475";

const API\_KEY\_TRANSLATE = "trnsl.1.1.20190509T193515Z.32ea28aa6befc900.b8f70d7ed8736e7b6613f94357cef16cb148038d"

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

segmentsInfo: false,

segmentsInfoTransfer: false,

error: undefined,

fromError: undefined,

toError: undefined

}

}

getCorrectCity(cityName){

var newCityName = "";

for(let i of cityName){

if(i === ' ' || i === '-'){

continue;

}

else{

newCityName += i;

}

}

return newCityName;

}

getSchedule = async (event) => {

event.preventDefault();

var fromCityText = event.target.elements.fromCity.value;

var toCityText = event.target.elements.toCity.value;

const allDate = event.target.elements.allDate.value;

const transferCheck = event.target.elements.transfer.checked;

var langOfSchedule = "en";

if (fromCityText){

var api\_url\_translate = await fetch(`https://translate.yandex.net/api/v1.5/tr.json/translate?key=${API\_KEY\_TRANSLATE}&text=${fromCityText}&lang=en`)

var dataOfTranslateText = await api\_url\_translate.json();

console.log(dataOfTranslateText);

fromCityText = dataOfTranslateText.text[0];

if(dataOfTranslateText.lang.substr(0, 2) === "ru"){

langOfSchedule = dataOfTranslateText.lang.substr(0, 2);

}

this.setState({

error: undefined,

fromError: undefined

});

}

else{

this.setState({

segmentsInfo: false,

segmentsInfoTransfer: false,

error: undefined,

fromError: "Введите город отправления"

});

}

if (toCityText){

api\_url\_translate = await fetch(`https://translate.yandex.net/api/v1.5/tr.json/translate?key=${API\_KEY\_TRANSLATE}&text=${toCityText}&lang=en`)

dataOfTranslateText = await api\_url\_translate.json();

toCityText = dataOfTranslateText.text[0];

console.log(dataOfTranslateText);

this.setState({

error: undefined,

toError: undefined

});

}

else{

this.setState({

error: undefined,

segmentsInfo: false,

segmentsInfoTransfer: false,

toError: "Введите город прибытия"

});

}

const api\_url\_city = await fetch(`https://iatacodes.org/api/v6/cities?api\_key=${API\_KEY\_CITY\_CODE}`)

const dataOfCity = await api\_url\_city.json();

var fromCityCode;

var toCityCode;

for (let i in dataOfCity.response) {

if (this.getCorrectCity(dataOfCity.response[i].name).toLowerCase() === this.getCorrectCity(fromCityText).toLowerCase()) {

fromCityCode = dataOfCity.response[i].code;

break;

}

}

for (let i in dataOfCity.response) {

if (this.getCorrectCity(dataOfCity.response[i].name).toLowerCase() === this.getCorrectCity(toCityText).toLowerCase()) {

toCityCode = dataOfCity.response[i].code;

break;

}

}

console.log(fromCityCode, toCityCode);

if (fromCityCode === undefined || toCityCode === undefined){

if(fromCityCode === undefined && this.state.fromError === undefined){

this.setState({

segmentsInfo: false,

segmentsInfoTransfer: false,

error: undefined,

fromError: "Введите корректный город"

});

}

if(toCityCode === undefined && this.state.toError === undefined){

this.setState({

segmentsInfo: false,

segmentsInfoTransfer: false,

error: undefined,

toError: "Введите корректный город"

});

}

}

else if(fromCityCode === toCityCode){

this.setState({

segmentsInfo: false,

error: "Города вылета и прибытия должны отличаться",

toError: undefined,

fromError: undefined

});

}

else{

const api\_url = await fetch(`http://api.rasp.yandex.net/v3.0/search/?apikey=${API\_KEY\_SCHEDULE}&format=json&from=${fromCityCode}&to=${toCityCode}&lang=${langOfSchedule}\_RU&page=1&date=${allDate}&transport\_types=plane&system=iata&transfers=${transferCheck}`);

const data = await api\_url.json();

console.log(data);

if(!transferCheck){

if (data.pagination.total === 0){

this.setState({

segmentsInfo: false,

segmentsInfoTransfer: false,

error: "Выберите поиск рейсов с пересадкой"

});

}

else{

this.setState({

segmentsInfo: data.segments,

segmentsInfoTransfer: false,

error: undefined,

fromError: undefined,

toError: undefined

});

}

}

else{

if (data.pagination.total === 0){

this.setState({

segmentsInfo: false,

segmentsInfoTransfer: false,

error: "Между этими городами не осуществляются авиаперелёты"

});

}

else if (!data.segments[0].has\_transfers){

this.setState({

segmentsInfo: false,

segmentsInfoTransfer: false,

error: "Доступны прямые рейсы"

});

}

else{

this.setState({

segmentsInfo: false,

segmentsInfoTransfer: data.segments,

error: undefined,

fromError: undefined,

toError: undefined

});

}

console.log(data);

}

}

}

render() {

return (

<div className="wrapper">

<div className="main">

<div className="container">

<div className="row justify-content-center">

<div className="info">

<Info />

</div>

<div className="form">

<Form scheduleMethod = {this.getSchedule} />

<ErrorInfo

fromError = {this.state.fromError}

toError = {this.state.toError}

error = {this.state.error}

/>

</div>

<div className="table">

<Schedule segmentsInfo = {this.state.segmentsInfo} />

<ScheduleTransfer segmentsInfoTransfer = {this.state.segmentsInfoTransfer} />

</div>

</div>

</div>

</div>

</div>

);

}

}

export default App;

**Info.js**

import React from "react";

class Info extends React.Component{

render(){

return(

<div>

<h2>Расписание авиарейсов</h2>

<p>Удобный сервис - быстрые путешествия!</p>

</div>

);

}

}

export default Info;

**Form.js**

import React from "react";

class Form extends React.Component{

constructor(props) {

super(props);

this.state = {

fromCity: 'Minsk',

toCity: 'Oslo',

allDate: '',

minDate: '',

maxDate: '',

isCheck: true

};

this.reverseCity = this.reverseCity.bind(this);

}

componentWillMount() {

const date = new Date();

const day = date.getDate();

const month = (date.getMonth() > 9) ? date.getMonth()+1 : "0" + (date.getMonth()+1);

const year = date.getFullYear();

const dateMax = new Date(year, +month + 2, day)

const maxDay = dateMax.getDate();

const maxMonth = (dateMax.getMonth() > 9) ? dateMax.getMonth()+1 : "0" + (dateMax.getMonth()+1);

const maxYear = dateMax.getFullYear();

this.setState({

allDate: `${year}-${month}-${day}`,

minDate: `${year}-${month}-${day}`,

maxDate: `${maxYear}-${maxMonth}-${maxDay}`

})

}

reverseCity(){

const nowFromCity = this.state.fromCity;

const nowToCity = this.state.toCity;

this.setState({

fromCity: nowToCity,

toCity: nowFromCity

});

}

handleChange = e => {

e.preventDefault();

this.setState({

[e.target.name]: e.target.value

})

}

handleChangeCheckbox = e => {

this.setState({

isCheck: !this.state.isCheck,

});

}

render(){

const { fromCity, toCity, allDate, minDate, maxDate, isCheck } = this.state;

return(

<form onSubmit={this.props.scheduleMethod} className="flex">

<input type="text" name="fromCity" value={fromCity} onChange={this.handleChange} placeholder="Город отправления"/>

<input type="button" onClick={this.reverseCity}/>

<input type="text" name="toCity" value={toCity} onChange={this.handleChange} placeholder="Город прибытия"/>

<input type="date" name="allDate" value={allDate} onChange={this.handleChange} max={maxDate} min={minDate}/>

<button>Показать рейсы</button>

<label className="checkbox">

<input type="checkbox" defaultChecked={isCheck} onChange={this.handleChangeCheckbox} name="transfer"/>

<div className="checkbox\_\_text">

Выводить рейсы с пересадками

</div>

</label>

</form>

);

}

}

export default Form;

**Schedule.js**

import React from "react";

class Schedule extends React.Component {

renderTable =() => {

let row;

if(this.props.segmentsInfo){

row = this.props.segmentsInfo.map((item, index) => (

<tr key={index}>

<td>{item.thread.number}</td>

<td>{item.thread.vehicle}</td>

<td>{item.from.title} - {item.to.title}</td>

<td>{item.departure.substr(11, 8) + ' ' + item.departure.substr(19, 6)}</td>

<td>{item.arrival.substr(11, 8) + ' ' + item.arrival.substr(19, 6)}</td>

</tr>

))

return(<table>

<tbody>

<tr>

<th>Номер рейса</th>

<th>Самолёт</th>

<th>Маршрут</th>

<th>Время вылета</th>

<th>Время прибытия</th>

</tr>

{row}

</tbody>

</table>)

}

}

render() {

console.log(this.props.segmentsInfo)

return ( <div >

{this.renderTable()}

</div>

);

}

}

export default Schedule;

**ScheduleTransfer.js**

import React from "react";

class ScheduleTransfer extends React.Component {

renderTable =() => {

let number, planeName, route, departureTime, arrivalTime;

if(this.props.segmentsInfoTransfer){

number = this.props.segmentsInfoTransfer.map((item, index) => (

<tr key={index}>

<td>

<tr>With transfer</tr>

<tr>{item.details[0].thread.number}</tr>

<tr>{item.details[2].thread.number}</tr>

</td>

</tr>

))

planeName = this.props.segmentsInfoTransfer.map((item, index) => (

<tr key={index}>

<td >

<tr>With transfer</tr>

<tr>{item.details[0].thread.vehicle}</tr>

<tr>{item.details[2].thread.vehicle}</tr>

</td>

</tr>

))

route = this.props.segmentsInfoTransfer.map((item, index) => (

<tr key={index}>

<td >

<tr>{item.departure\_from.title} - {item.arrival\_to.title}</tr>

<tr>{item.details[0].from.title} - {item.details[0].to.title}</tr>

<tr>{item.details[2].from.title} - {item.details[2].to.title}</tr>

</td>

</tr>

))

departureTime = this.props.segmentsInfoTransfer.map((item, index) => (

<tr key={index}>

<td >

<tr>{item.departure.substr(11, 8) + ' ' + item.departure.substr(19, 6)}</tr>

<tr>{item.details[0].departure.substr(11, 8) + ' ' + item.departure.substr(19, 6)}</tr>

<tr>{item.details[2].departure.substr(11, 8) + ' ' + item.departure.substr(19, 6)}</tr>

</td>

</tr>

))

arrivalTime = this.props.segmentsInfoTransfer.map((item, index) => (

<tr key={index}>

<td >

<tr>{item.arrival.substr(11, 8) + ' ' + item.arrival.substr(19, 6)}</tr>

<tr>{item.details[0].arrival.substr(11, 8) + ' ' + item.arrival.substr(19, 6)}</tr>

<tr>{item.details[2].arrival.substr(11, 8) + ' ' + item.arrival.substr(19, 6)}</tr>

</td>

</tr>

))

return(<div className="table-responsive"><table className="table">

<tbody>

<tr>

<th>Номер рейса</th>

<th>Самолёт</th>

<th>Маршрут</th>

<th>Время вылета</th>

<th>Время прибытия</th>

</tr>

<td>{number}</td>

<td>{planeName}</td>

<td>{route}</td>

<td>{departureTime}</td>

<td>{arrivalTime}</td>

</tbody>

</table>

</div>)

}

}

render() {

console.log(this.props.segmentsInfoTransfer)

return ( <div >

{this.renderTable()}

</div>

);

}

}

export default ScheduleTransfer;

**ErrorInfo.js**

import React from "react";

class ErrorInfo extends React.Component{

render(){

return(

<div className="errors">

<span className="error">{this.props.fromError}</span>

<span className="error1">{this.props.toError}</span>

<span className="error2">{this.props.error}</span>

</div>

);

}

}

export default ErrorInfo;

**App.css**

@import url('https://fonts.googleapis.com/css?family=PT+Sans+Caption');

html{

height: 100%;

}

table {

font-size: 14px;

background: white;

max-width: 100%;

width: 100%;

border-collapse: collapse;

text-align: left;

}

th {

font-weight: normal;

color: #039;

border-bottom: 2px solid #113a65;

padding: 10px 8px;

}

td {

color: #113a65;

padding: 9px 8px;

transition: .3s linear;

}

tr:hover td {

color: #6699ff;

}

.wrapper {

font-family: "Lucida Sans Unicode", "Lucida Grande", Sans-Serif;

padding: 70px 0;

min-height: 100vh;

display: flex;

justify-content: center;

align-items: flex-start;

background: rgba(135, 237, 232, 1);

background: -moz-linear-gradient(-45deg, rgba(135, 237, 232, 1) 0%, rgba(106, 235, 173, 1) 100%);

background: -webkit-gradient(left top, right bottom, color-stop(0%, rgba(135, 237, 232, 1)), color-stop(100%, rgba(106, 235, 173, 1)));

background: -webkit-linear-gradient(-45deg, rgba(135, 237, 232, 1) 0%, rgba(106, 235, 173, 1) 100%);

background: -o-linear-gradient(-45deg, rgba(135, 237, 232, 1) 0%, rgba(106, 235, 173, 1) 100%);

background: -ms-linear-gradient(-45deg, rgba(135, 237, 232, 1) 0%, rgba(106, 235, 173, 1) 100%);

background: linear-gradient(135deg, rgba(135, 237, 232, 1) 0%, rgba(106, 235, 173, 1) 100%);

filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#87ede8', endColorstr='#6aebad', GradientType=1);

}

.main {

height: 100%;

background: #51a4fd57;

box-shadow: 0px 2px 80px -20px rgba(0, 0, 0, 0.5);

width: 80%;

margin: 0 auto;

}

.checkbox {

margin: 10px;

}

.checkbox input {

position: absolute;

z-index: -1;

opacity: 0;

}

.checkbox\_\_text {

position: relative;

padding: 0 0 0 60px;

cursor: pointer;

flex: 100%;

}

.checkbox\_\_text:before {

content: '';

position: absolute;

top: -1px;

left: 0;

width: 50px;

height: 26px;

border-radius: 13px;

background: #CDD1DA;

box-shadow: inset 0 2px 3px rgba(0, 0, 0, .2);

transition: .2s;

}

.checkbox\_\_text:after {

content: '';

position: absolute;

top: 1px;

left: 2px;

width: 22px;

height: 22px;

border-radius: 10px;

background: #FFF;

box-shadow: 0 2px 5px rgba(0, 0, 0, .3);

transition: .2s;

}

.checkbox input:checked+.checkbox\_\_text:before {

background: #113a65;

}

.checkbox input:checked+.checkbox\_\_text:after {

left: 26px;

}

input[type="text"] {

background-color: white;

border: 0;

border: solid 2px #113a65;

width: 25%;

padding: 4px;

color: #113a65;

height: 42px;

font-weight: lighter;

margin: 20px 10px;

outline: none;

}

input[type="button"] {

border: 1px solid rgba(0, 0, 0, 0);

outline: none;

padding: 20px;

height: 20px;

margin: 5px;

margin-left: 3px;

background: url(https://www.shareicon.net/data/2016/05/24/770231\_arrows\_512x512.png) no-repeat 3px 1px;

background-size: auto 90%;

}

[type="date"] {

background: #fff url(https://cdn1.iconfinder.com/data/icons/cc\_mono\_icon\_set/blacks/16x16/calendar\_2.png) 97% 50% no-repeat;

border: 0;

border: solid 2px #113a65;

width: 20%;

height: 42px;

padding: 3px;

color: #113a65;

font-weight: lighter;

margin: 20px 10px;

outline: none;

}

[type="date"]::-webkit-inner-spin-button {

display: none;

}

[type="date"]::-webkit-clear-button {

display: none;

}

[type="date"]::-webkit-calendar-picker-indicator {

opacity: 0;

}

button {

background-color: white;

outline: none;

color: black;

border: 2px solid #113a65;

padding: 7px 15px;

}

button:hover {

background-color: #113a65;

color: white;

}

.info{

font-family: PT Sans Caption;

text-align: center;

width: 90%;

height: 345px;

background-image: url(http://clipart-library.com/img/1152812.png);

background-blend-mode: multiply;

background-color: rgba(255, 255, 255, 0);

background-size: cover;

}

.info h2{

color: #039;

font-size: 2.3em;

text-shadow: #039 -8px 4px 8px;

margin-left: 120px;

margin-top: 30px;

}

.info p{

color: #039;

font-size: 1.5em;

text-shadow: #039 -3px 3px 10px;

margin-left: 353px;

margin-top: 5px

}

.form{

margin-left: 45px;

}

.flex {

display: flex;

align-items: center;

width: 100%;

flex-wrap: wrap;

}

.errors {

position: relative;

}

.error {

position: absolute;

color: red;

top: -65px;

left: 10px;

}

.error1 {

position: absolute;

color: red;

top: -65px;

left: 354px;

}

.error2 {

position: absolute;

color: red;

top: -61px;

left: 10px;

}