Introduction:

This document is highlighting the essential technical stack needed to deploy the project on staging or production servers. It also details out the necessary steps to project source code on development environment to initial development directory.

Apart for setup and deployment processes this report also presents in-depth technological requirement to execute this project which is a useful guide for any development to fork and kick-off development cycles.

Software requirements:

1. Need adequate version of Node.js. Can check the version using command prompt.

```
$ node --version
v6.10.1
```

2. Compactable version of npm with node.js. It can be cloneable using:

```
$ npm --version
3.10.10
```

3. **(Development only)** Need an Integrated development environment (IDE) or code editor for development purposed. There are two options which are as follows:

    a. Write Code in the Browser
    b. Local Development Environment (like: Jet brain, NetBeans or notepad++)

4. **(Deployment only)** Node static server to serve the application on local or development server for testing.

```
npm install -g serve
serve -s build
```

Process to configure project source code:

To start a development and testing on local system user are going to require all the software listed out in software requirements above.

System configuration:

1. Make sure node is in the system by checking the node version from command prompt.

C:> node -v

or

C:> node -version

2. Make sure npm is installed on local system. The easy way of doing this is check npm version from command prompt.

C:> npm -v

or

C:> npm -version

Process to clone:

1. **Copy source code and make directory structure:**

   Copy project source code in development directory. Make sure the directory structure should be in accordance to React JS directory standard to avoid any build and run issue and it will also enhance code readability which in return help in code maintainers.

   Local directory structure can use the following example as a guide to setup source code directory.

```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
```

2. **Install npm package:**

To install npm packages, you need to go in project directory using command prompt example:

```
zObY@zObY-PC MINGW64 /d/University/ICT90004
$ cd leasepass/
```

Once you are in project directory check for package.json file in project parent folder.

```
zObY@zObY-PC MINGW64 /d/University/ICT90004/leasepass
$ dir
README.md  build  node_modules  package-lock.json  package.json  public  src
```

package.json file contain all the reactJS components used by our project to run and enable development we need to install all the packages, before installing check the package.json.

To do that open package.json file in an editor and check the file should is in line with the following file snippets. Make sure all the components are there with proper version as shows below.

```
"name": "leasepass",
"version": "0.1.0",
"private": true,
"dependencies": {
  "bootstrap": "^4.1.3",
  "font-awesome": "^4.7.0",
  "google-map-react": "^1.0.9",
  "jquery": "^3.3.1",
  "rc-slider": "^8.6.3",
  "react": "^16.5.2",
  "react-autocomplete": "^1.8.1",
  "react-datalist-input": "^1.0.4",
  "react-dom": "^16.5.2",
  "react-loader-spinner": "^2.0.6",
  "react-rangeslider": "^2.2.0",
  "react-responsive-carousel": "^3.1.43",
  "react-router-dom": "^4.3.1",
  "react-router-hash-link": "^1.2.0",
  "react-scripts": "2.0.4",
  "react-spinners": "^0.4.5"
},
```

Once everything is checkout, now install all the packages using following command.

```
zObY@zObY-PC MINGW64 /d/University/ICT90004/leasepass
$ npm install
```

The above command will install the necessary components and install package step is done.

3. **Start application development server:**

After completing above steps, we are ready to start our development server and live development on local system.

First step, in starting a development server is to make sure system port 3000 is empty. For that you can use following command:

`netstat -ano | findstr 3000`

```
zObY@zObY-PC MINGW64 /d/University/ICT90004/leasepass
$ netstat -ano | findstr 3000
```

If there is nothing return for above command, then port is available.

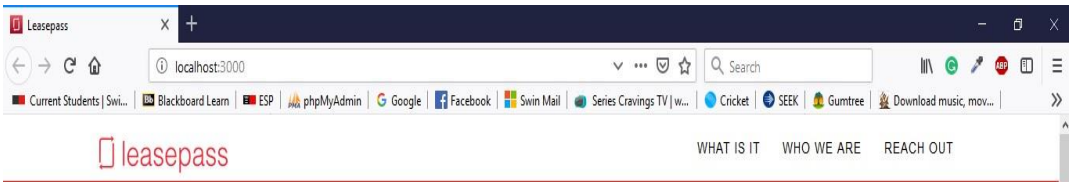
Now, we are ready to start development server on port 3000 by using following command.

```
zObY@zObY-PC MINGW64 /d/University/ICT90004/leasepass
$ npm start
```

Once it is successfully executing the application will open on the default system browser. And command prompt will display the following status and will act as logs.

```
Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.
```

The browser will be navigating to localhost:3000 like this.



Process to deploy project build:

To deploy this project, you need to install steps 1, 2 and 4 from software requirement. Along with project build folder which can be found under project source code if not then you can create a build using following command:

1. Go to project directory, as shows below.

```
zObY@zObY-PC MINGW64 /d/University/ICT90004
$ cd leasepass/
```

2. Execute npm run build command as shows below.

```
zObY@zObY-PC MINGW64 /d/University/ICT90004/leasepass
$ npm run build
```

3. The above build executing will generate a build folder under project directory and end with the following information on command prompt

```
File sizes after gzip:

  138.89 KB  build\static\js\1.508b7398.chunk.js
  27.88 KB   build\static\css\1.43b71680.chunk.css
  14.93 KB   build\static\js\main.5dea3480.chunk.js
  2.62 KB    build\static\css\main.ca1f71da.chunk.css
  763 B      build\static\js\runtime~main.229c360f.js

The project was built assuming it is hosted at the server root.
You can control this with the homepage field in your package.json.
For example, add this to build it for GitHub Pages:

  "homepage" : "http://myname.github.io/myapp",
```

Deploy on static server:

1. Once you have the build folder, you can move this folder to your desire location or on hosting or production server.

2. Now we can serve build folder on using node static server on production or staging. Using following command like

```
zObY@zObY-PC MINGW64 /d/University/ICT90004/leasepass
$ serve -s build -p 80
```

In above command -p 80 is identifying the port number we are hosting our application in this case we are using port 80.

3. Now your application is ready to be use by local and network traffic on following information.

```
Serving!

- Local:            http://localhost:80
- On Your Network:  http://192.168.60.2:80

Copied local address to clipboard!
```

Other deployment options:

Other suitable server deployment options are as following according to reactJS documentation.

Deployment with ExpressJS

You don't necessarily need a static server in order to run a Create React App project in production. It works just as fine integrated into an existing dynamic one.

Here's a programmatic example using Node and Express:

```
const express = require('express');
const path = require('path');
const app = express();

app.use(express.static(path.join(__dirname, 'build')));

app.get('/', function(req, res) {
  res.sendFile(path.join(__dirname, 'build', 'index.html'));
});

app.listen(9000);
```

The choice of your server software isn't important either. Since Create React App is completely platform-agnostic, there's no need to explicitly use Node.

Serving Apps with Client-Side Routing:

If you use routers that use the HTML5 pushState history API under the hood (for example, React Router with browserHistory), many static file servers will fail. For example, if you used React Router with a route for /todos/42, the development server will respond to localhost:3000/todos/42properly, but an Express serving a production build as above will not.

This is because when there is a fresh page load for a /todos/42, the server looks for the file build/todos/42 and does not find it. The server needs to be configured to respond to a request to /todos/42 by serving index.html. For example, we can amend our Express example above to serve index.html for any unknown paths:

```
 app.use(express.static(path.join(__dirname, 'build')));

-app.get('/', function (req, res) {
+app.get('/*', function (req, res) {
   res.sendFile(path.join(__dirname, 'build', 'index.html'));
```

```
});
```

If you're using Apache HTTP Server, you need to create a .htaccess file in the public folder that looks like this:

```
Options -MultiViews
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.html [QSA,L]
```

It will get copied to the build folder when you run npm run build.

If you're using Apache Tomcat, you need to follow this Stack Overflow answer.

Now requests to /todos/42 will be handled correctly both in development and in production.

On a production build, and when you've opted-in, a service worker will automatically handle all navigation requests, like for /todos/42, by serving the cached copy of your index.html. This service worker navigation routing can be configured or disabled by ejecting and then modifying

thenavigateFallback and navigateFallbackWhitelist options of the SWPreachePlugin configuration.

When users install your app to the homescreen of their device the default configuration will make a shortcut to /index.html. This may not work for client-side routers which expect the app to be served from /. Edit the web app manifest at public/manifest.json and change start_url to match the required URL scheme, for example:

```
"start_url": ".",
```

References:

Anon 2018, Deployment · Create React App, viewed 25 October, 2018, <https://facebook.github.io/create-react-app/docs/deployment>.

Anon 2018, *Tutorial: Intro to React – React*, viewed 25 October, 2018, <https://reactjs.org/tutorial/tutorial.html>.

Anon 2018, facebook/create-react-app, viewed 25 October, 2018, <https://github.com/facebook/create-react-app>.