Deployment Documentation for React Application

# 1. Pre-requisites

Before deploying the application, ensure the following:  
- Node.js is installed on your system. You can download it from nodejs.org.  
- npm (Node Package Manager) comes bundled with Node.js. It's used to install and manage packages required by your project.  
- Your project files are located in the correct directory.  
- Ensure access to the server where you plan to deploy the application (Windows or Linux).  
  
Additional software versions to ensure:  
- Node.js version: v18.12.0  
- npm version: 9.9.3

# 2. Environment Setup

Your project uses different environments such as development (dev), system integration testing (SIT), and production (prod). For each environment, specific settings are stored in environment files (.dev.env, .sit.env, .prod.env). These files hold sensitive information like API keys, database URLs, and configuration settings for each environment.

## 2.1 How to set up environment files:

- .dev.env: Used for development.  
- .sit.env: Used for system integration testing.  
- .prod.env: Used for production.  
  
These files should be placed in the root folder of your project. Each environment will load different configurations when you deploy the application.

# 3. Build Commands

The project uses several commands to build and start the application in different environments:

## 3.1 Available Scripts

- npm run build: Creates a production-ready build of your application.  
- npm run start: Starts the development server to preview the application.  
- npm run build:dev: Creates a build specifically for the development environment by loading settings from .dev.env.  
- npm run build:sit: Creates a build specifically for the SIT environment by loading settings from .sit.env.  
- npm run build:prod: Creates a production build by loading settings from .prod.env.

## 3.2 Build Process Explained:

1. Start Development Server: To start working locally, run:  
 ```bash  
 npm run start  
 ```  
 This command uses React Scripts to start a local development server, enabling you to preview and test the app on localhost:3000.  
  
2. Build for Production: When you're ready to deploy, you need to build the project:  
 ```bash  
 npm run build  
 ```  
 This command creates optimized files in the build folder, which are ready for deployment to your server.  
  
3. Build for Different Environments: To build for a specific environment (e.g., development, SIT, or production), use:  
 ```bash  
 npm run build:dev # For development build  
 npm run build:sit # For SIT build  
 npm run build:prod # For production build  
 ```

# 4. Post Build Process

Once the build is complete, some additional steps are required to prepare your project for deployment.

## 4.1 Web.xml Configuration (Java Servlets Integration)

After building the application, the postbuild command will:  
- Create a directory called WEB-INF inside the build folder.  
- Copy the web.xml file into this directory to ensure proper configuration for integration with Java-based servers.  
  
The command used is:  
```bash  
mkdirp build/WEB-INF && copyfiles web.xml build/WEB-INF  
```

# 5. Deploying to Servers

## 5.1 Deploying to Windows Server

If you're deploying to a Windows server, follow these steps:  
  
1. Copy files: Use the following command to copy the build files to the desired directory on the Windows server (in this case, to a Tomcat web server directory):  
 ```bash  
 npm run deploy:windows  
 ```  
 This command will copy all files from the build folder to C:\Workspace\software\apache-tomcat-9.0.95\webapps\audit-manager\.  
  
2. Restart Tomcat: After copying the files, restart the Tomcat server to see the updated application.

## 5.2 Deploying to Linux Server

If you're deploying to a Linux server, follow these steps:  
  
1. Copy files: Use the following command to copy the build files to the desired directory on the Linux server (in this case, to an Apache server directory):  
 ```bash  
 npm run deploy:linux  
 ```  
 This command will copy all files from the build folder to /var/www/html/audit-manager/.  
  
2. Restart Apache: After copying the files, restart the Apache server to see the updated application. You can restart Apache using the command:  
 ```bash  
 sudo systemctl restart apache2  
 ```

# 6. Cleaning the Project and Starting Fresh

If you face any issues, you can clean your project and reinstall all dependencies using the following command:  
```bash  
npm run install:clean  
```  
This command will:  
- Remove the node\_modules folder and package-lock.json file.  
- Reinstall all the necessary project dependencies.  
- Automatically start the development server.

# 7. Conclusion

By following these steps, you can easily deploy your React application to both Windows and Linux servers. Be sure to choose the correct environment and server type, and carefully follow the commands outlined above.  
  
If you encounter any issues or need further assistance, feel free to contact the technical team.

# 8. Making REST Calls Using Axios

Axios is used in this project for making HTTP requests. It allows you to make requests to external APIs or internal endpoints to retrieve or send data.  
  
Example of a GET request using Axios:  
```javascript  
import axios from 'axios';  
  
axios.get('https://api.example.com/data')  
 .then(response => {  
 console.log(response.data);  
 })  
 .catch(error => {  
 console.error('Error fetching data:', error);  
 });  
```  
  
Example of a POST request:  
```javascript  
axios.post('https://api.example.com/data', {  
 key1: 'value1',  
 key2: 'value2'  
 })  
 .then(response => {  
 console.log('Data saved:', response.data);  
 })  
 .catch(error => {  
 console.error('Error saving data:', error);  
 });  
```  
These Axios examples show how to interact with APIs in your application.